Information Management with Multiple-Categorization Hierarchical Structure

By

Yifeng Gao

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

Jan 30, 2006

© Copyright
2006, Yifeng Gao
NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Canada

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Abstract

Due to the explosion of information in modern computerized world, it has become more and more important to efficiently handle huge volume of information and existing systems available for managing such information have become outdated. CASH [44] has proposed a category-based solution to overcome the drawbacks of existing directory-based systems to manage information. However it still has many limitations and opening issues. In this thesis, we mainly describe our work, an information management system based on multiple-categorization hierarchical structure, which is an extension to CASH. We have not only improved the drawbacks and limitations of CASH, but also enhanced and proposed a context-based search based on category structure. We have also evaluated our system by mapping both UNIX file system and ODP structure dump, and done comparisons between our system and other related applications to prove that our system provides users a more flexible and efficient way to manage information.
Acknowledgements

First and foremost, I would like to express my sincere thanks to my supervisor Dr. Mengchi Liu, for being such a wonderful advisor. Without him, this thesis would not have been possible. I thank him for his patience and encouragement that carried me on through difficult times, and for his insights and suggestions that helped to shape my research skills. His valuable feedback contributed greatly to this thesis.

I cannot fully express my gratitude to my parents and wife. We always feel so close, though they live at the other end of the earth. Their encouragement and well-being are the biggest supports for me.

My gratitude also to Peng Lu for offering server support during the implementation and to Zhihong Li and Qunxiao Wang for sharing ideas with me during the research.
Contents

Abstract iii
Acknowledgements iv
List of Tables viii
List of Figures x

1 Introduction 1
  1.1 Motivation ............................................................... 1
  1.2 Objectives ............................................................... 5
  1.3 Outline of thesis ...................................................... 7

2 Background 9
  2.1 Directory-based applications ................................. 9
    2.1.1 Directory ......................................................... 9
    2.1.2 File systems ..................................................... 10
    2.1.3 Directory-based search engines ......................... 12
    2.1.4 Online posting systems .................................... 13
    2.1.5 Other directory-based applications on desktop .... 14
    2.1.6 Summary .......................................................... 15
6.3.3 Enhanced context-based search algorithm ............................................ 74

7 Experimental system ................................................................. 78
  7.1 Roles of the users ............................................................................................ 78
  7.2 Search types .................................................................................................... 79
  7.3 Category manipulation ................................................................................... 80
  7.4 Multiple search solutions .............................................................................. 88
    7.4.1 Manually dig deeper .............................................................................. 88
    7.4.2 Context-based searches ......................................................................... 89
    7.4.3 Summary ............................................................................................... 90
  7.5 Experiment and comparison ........................................................................ 91
    7.5.1 UNIX file system mapping ....................................................................... 91
    7.5.2 ODP structure dump mapping .................................................................. 93
    7.5.3 Feature-based comparison ...................................................................... 95
    7.5.4 Performance-based comparison ............................................................. 102

8 Conclusion remarks and future work ...................................................... 104

Bibliography ............................................................................................. 107
List of Tables

1.1 Search Cases .............................................................. 2
5.1 Example of Category ................................................... 55
5.2 Example of Category classification ............................... 56
5.3 Example of Virtual category classification .................... 57
5.4 Example of Document .................................................. 58
5.5 Example of Document classification .............................. 59
5.6 Example of URL .......................................................... 60
5.7 Table Category classification after adding category .......... 62
5.8 Table Category after adding category ............................. 62
5.9 Table Category classification after removing category without offsprings ............................. 63
5.10 Table Category after removing category without offsprings .......... 64
7.1 Comparison ............................................................... 100
7.2 Comparison (continue1) ............................................... 100
7.3 Comparison (continue2) ............................................... 101
7.4 Search performance comparison .................................... 102
List of Figures

2.1 Links in UNIX file systems and directory-based search engines .......... 12

4.1 Category Structure ................................................................................. 34

5.1 Struts framework ...................................................................................... 44
5.2 System architecture ................................................................................... 45
5.3 Entity-Relationship Model Diagram ......................................................... 51
5.4 Database Schema ....................................................................................... 52
5.5 Sample Multiple-categorization hierarchical structure ............................. 54
5.6 Sample Revised Multiple-categorization hierarchical structure ............... 61
5.7 Move operation scenarios ........................................................................ 66
5.8 Algorithm of move operation .................................................................. 67
6.1 Context-based search ............................................................................... 71
6.2 Algorithm of path computing ................................................................. 74
6.3 Algorithm of enhanced context-based search .......................................... 76
6.4 Enhanced context-based search steps ..................................................... 77
7.1 Administrator Console ............................................................................. 79
7.2 User Console ............................................................................................. 80
7.3 Search main page ...................................................................................... 81
7.4 Sample structure ...................................................................................... 82
7.5 Move category ................................................................. 83
7.6 Remove category ............................................................. 83
7.7 Edit categorization relationship ........................................ 84
7.8 Main category search display after adding logical subcategory ........ 85
7.9 Upload and categorize documents ........................................ 86
7.10 Revised sample structure .................................................. 87
7.11 Manual search ............................................................... 88
7.12 Search for Java and display the result in wrapped category ........ 89
7.13 Result of wrapped category ................................................ 89
7.14 Algorithm: Search under a certain category ......................... 90
7.15 Algorithm: UNIX file system mapping ............................... 92
7.16 Mapping between ODP RDF structure and links on the web page ........ 93
7.17 ODP mapping algorithm .................................................... 94
7.18 Category of Games ........................................................ 95
7.19 Web Category based Unix File System ............................... 96
7.20 Category of ygao ............................................................. 97
7.21 Category of www ........................................................... 98
7.22 Web Category search ....................................................... 99
7.23 Search performance comparison ......................................... 103
Chapter 1

Introduction

1.1 Motivation

Electronic information is one of the most important atomic elements and core concepts in modern computerized world. Due to the explosion of information in the computerized world within the last several decades, it has become more and more important to manage tremendous volume of information. Electronic information concerns almost every aspect within the computerized world, from electronic documents or files on PCs or servers to electronic information within search engines. How to organize, manage, maintain, and search for huge volume of electronic information has always been a topic and challenge to computer scientists. Directory structure, which is a tree structure, has been the most widespread structure so far to handle this job. Most of the corresponding applications, such as file systems, email management systems, directory-based search engines, online posting systems and etc., are all based on the directory structure. Although this hierarchical structure does simplify the information management, it has obvious drawbacks and limitations, which lead to the bottleneck of managing the information in a flexible and efficient way. Here we list some scenarios, which are daily behaviors in computerized world and may concern computer users everyday. In order to make it more presentive, we make the
use cases according to the behaviors on search engines, to which other information management applications such as file systems, email management systems or online posting systems are similar. Here is the list of the use cases.

<table>
<thead>
<tr>
<th>Search Cases</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Java (the island in Asia)</em></td>
<td>Some keywords are unevenly distributed online. This case shows the disadvantage of</td>
</tr>
<tr>
<td></td>
<td>keyword-based search</td>
</tr>
<tr>
<td><em>NANO (a type of MP3 player from Apple)</em></td>
<td>This case shows the disadvantage of using directory-based search manually.</td>
</tr>
<tr>
<td></td>
<td>Combination of directory-based and keyword-based searching is an advantage for this</td>
</tr>
<tr>
<td></td>
<td>case</td>
</tr>
<tr>
<td><em>SCS Carleton University</em></td>
<td>Simple searching case doesn’t work under combination of directory-based and</td>
</tr>
<tr>
<td></td>
<td>keyword-based searching. This case shows drawbacks of the directory structure.</td>
</tr>
</tbody>
</table>

**Case 1—Java**  
Java is one of the most popular buzzwords online today and generally people consider it as a programming language. However Java actually has other meanings such as an island in Asia, a slang for coffee and etc. Due to the page ranking algorithm of Google, which ranks web pages according to the link-based algorithm [35], that is, the more popular the web page is, the higher rank it will get in Google’s database, the web pages about Java programming language is obviously flooding over the internet and overwhelming the limited pages, which offer tourism information on Java Island. Consequently, Java programming pages occupy most of the positions within the searching results, especially in the front positions. By inputting *Java* in Google keyword web search, we get

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
about 149,000,000 matching results within a second. The next problem is how to dig out what we are interested in. It is hard to locate the desired Java programming related target from so many results, let alone to get the traveling information of an island in Asia. Then we have to be suffered from searching for the minority of Java island information from the large number of irrelevant results. In this case we cannot find the target information within the first 10 pages of results. This is far beyond an ordinary user’s patience. On the contrary, in this case, if we use the directory-based searching, like Google Directory or Yahoo Directory, we can easily find the corresponding information, which is well categorized, by some digging operations deeper into the directory structure. For instance, we can find it in Google Directory along this path: Regional → Asia → Indonesia → Provinces while get it in Yahoo Directory along this path: Directory → Regional → Countries → Indonesia → Provinces, Regions, and Districts

Case 2—NANO In this case we are trying to find NANO, a type of MP3 player from Apple. If we use directory-based search engine to find the target information, for example we apply Yahoo Directory, we may finally approach our destination along the path of Directory → Entertainment → Consumer Electronics → Audio → MP3 Players → Product Information and Reviews → Apple iPod → Apple iPod nano. There are around eight levels for us to explore by digging operations deeper into the directory structure and within each level there are quite a lot of directories for us to pick up the right option to approach our target. This obviously brings inconvenience to users who want to locate the targets which are very deep in the directory. In this case, we can apply the function of searching targets under a certain directory such as searching under the directory of Entertainment. By inputting the keyword of NANO and searching under Entertainment we can easily approach our target instead of the cumbersome digging deeper method.
Case 3—SCS Carleton University Based on the searching cases above, we find that a combined searching solution, which takes the advantages of both keyword-based and directory-based searching, is more efficient and accurate than any of these two methods used alone. Therefore in this case, we apply the combined searching solution to find the information on School of Computer Science in Carleton University. We may search the keyword *SCS Carleton U.* under the directory *Education* (it’s common to think of *Education* as searching space if we want to find something about schools). Unfortunately, we don’t hit the target this time. By exploring deeper into the directory structure we find the information of *SCS Carleton U.* is under the directory of *Computers → Computer Science → Academic Departments → North America → Canada* and there’s no way for us to find it under the directory *Education* based on the directory structure. Here are the reasons:

- A directory structure is a tree structure, in which every directory has only one parent and there is only one path from the root for each directory.
- Every document can only be placed in a certain directory.
- Every document has only one path from the root so that it could not be noted of all of its ancestors, which include both its ancestor directories and virtual ancestor directories from the links.

These are the drawbacks and limitations of widespread directory structures. We are not challenging the performance and searching features of Google here. In Google, there are many searching tools and features for advanced search solutions, such as searching within an organization, though most of them are not frequently applied by ordinary users. We use the samples above to check the flexibility and performance of the directory structure (ODP), applied by Google. In addition to the search engine cases above, actually this directory structure problem prevalently exists in all of the directory based applications listed below.

- File systems: UNIX, LINUX and Windows file systems apply directory structure to manage the files. In addition to the limited way to classify files, file information
are stored as inode information within the file systems and the system applies a tree traversal every time when locating a target file. This not only hinders the performance of the system, but also affect the accuracy and efficiency of searching.

- Email management systems: Outlook Express, Elm, Pine, Mutt and etc. all apply folders to manage the emails. In these systems, emails are arranged in a flat or hierarchy tree structure.

- Bookmarks management within browsers: Firefox’s live bookmarks, Internet Explorer’s Favorites, Opera’s bookmarks and Mosaic’s Hotlist all apply directory structure to manage the web page bookmarks.

- Online forums and blog directory: Online posting systems, such as Java Developer Forums or blogcatolog [5], apply directory structure to manage the electronic postings online.

We discuss the problems of the existing systems separately in detail in the next chapter. Some improvements need to be done according the drawbacks and limitations of the directory structure listed above and this is what we intend to do in this thesis.

1.2 Objectives

The overall goal of the work presented in this thesis is to describe an information management system based on multiple-categorization hierarchical structure, which overcomes the drawbacks of the existing systems based on directory structure and offers more flexibility and efficiency for users to organize, manage, maintain, and search for electronic information. Here are the nice features of our information management system based on multiple-categorization hierarchical structure over existing directory-based systems:
• Our system provides more flexible and natural way for information classification. Every document or information can be categorized into more than one parent categories. Documents are organized in corresponding categories, which are also allowed to be multiply categorized. This offers our system more alternative ways to refine the classifications in different levels.

• Due to the multiple-categorization of documents or information, it takes less efforts for users to retrieve target documents or information from any aspects of interests.

• Every documents or information within our system are noted of all of their parent and ancestor categories. That enables our system a bottom-up searching solution, such as enhanced context-based search. We will discuss it in detail in chapter 6.

• Our system is portable and extensible. It provides wider scope of usage. We can apply our system not only for document management, such as file system, but also for other information management, such as directory-based search engines, online posting systems, Wikis and etc.

We mainly extend the work based on CASH [44] and we have made a lot of improvements and progress to CASH. Here is a brief list of our work separated from the work of CASH [44] and these are also the objectives of this thesis:

• CASH has limited survey on the related works of the research and we should do more research on the existing solutions and related works from both academic community and industry.

• CASH puts little efforts on formalizing the general concepts on the structure, which should be very important to understand and describe the idea in a consistent way throughout the thesis and we should emphasize in this aspect.
• CASH mainly extends the shell based on the UNIX file system, therefore it’s platform-dependent and can only apply on the UNIX file system. We should implement a more portable and platform-independent system.

• Although CASH is an improvement to the UNIX file system, it still possesses some drawbacks of UNIX file system, such as it could locate the target via an absolute path but could not get all the paths information via a target. This restricts the bottom-up searching method and could not fulfill users’ request for being noted of more information on a target [36]. We should further improve this aspect to attain more efficient and resourceful search solutions.

• Although CASH has done a lot on the manipulations of category structure, it is short of searching power, which should also be enhanced in our work.

• Although CASH proposed an idea to attain the search efficiency by B-tree index searching within the supplementary database system, it has been poorly modeled and designed and does not get full use of the nice features and advantages of database system. We should re-model and re-design the whole database schema to attain the ultimate goal.

• Although CASH describes a nice idea and system to solve the widespread problem in information management, it does not provide any experiments and evaluations to prove itself. We should do much more in this aspect.

1.3 Outline of thesis

The thesis has four parts. The first part introduces some background knowledge concerned in this thesis. The second part gives out the related work of our research. The third part explains our approaches in detail. The fourth part describes the implementation of our
information system based on multiple-categorization hierarchical structure and do some comparison between our system and other existing applications. The thesis ends with a summary, conclusion and future research questions. Here is the detailed list of the content in the latter chapters.

- chapter 2 presents some important background knowledge, which is important to understand the thesis. The introduction is concentrated on the aspects of the problems of current directory-based applications and context-based search.

- chapter 3 is a critical review of the related work of our research.

- chapter 4 explains the concepts and the features of multiple-categorization hierarchical structure.

- chapter 5 gives the model of our category structure, including the Entity-Relationship model, database schema and operation rules.

- chapter 6 introduces an enhanced context-based searching method based on our category structure.

- chapter 7 shows our information management system, which is a practical combination of the models and algorithms above. After showing some simple use cases applied on the system, we evaluate the system by mapping both UNIX file system and ODP structure dump [30] into our database and prove that the information management system based on multiple-categorization hierarchical structure can solve the problems within widespread directory-structure-based applications.

- chapter 8 presents the conclusions of this thesis and describes directions for future research.
Chapter 2

Background

This chapter gives a brief background introduction to some related aspects, such as the directory-based applications, the limitations and drawbacks caused by the directory structure for these applications and context-based search.

2.1 Directory-based applications

2.1.1 Directory

What is a directory? A directory is an entity which contains a group of files and other directories. A directory structure is typically a tree structure, in which every directory or file can only belong to one directory. A directory structure is helpful to organize thousands of files by keeping related files together. A directory contained inside another directory is called a subdirectory of that directory. Since a directory structure is a tree structure, the natures of tree structure do have some rigid restrictions on the structure. For instance, every directory can only have one parent directory, except for the root directory, which has no parent directory, and every file can only be placed under a certain directory.
2.1.2 File systems

In computerized world, a file system is the way in which files are named and where they are placed logically for storage and retrieval. The DOS, Windows, OS/2, Macintosh, and UNIX-based operating systems all have file systems in which files are placed somewhere in a hierarchical tree structure. A file is placed in a directory (folder in Windows) or subdirectory at the desired place in the tree structure [16].

Although the directory structure has been evolved from original flat structure to hierarchical tree structure, the restrictions of the tree structure still prevent directory structure from flexibility. For example, one has to figure out the only absolute path of a certain directory, in which the target file is placed, to find a file. Therefore, file systems such as UNIX file system proposed extra features based on the directory structure to offer more flexible usage. The most important and widespread usage is the links within the directory structure.

There are two kinds of links within UNIX file system: hard link and symbolic link.

**Hard link**  It is a reference, or pointer, to the physical data. On most file systems, all named files are hard links. The name associated with the file is simply a label that refers the operating system to the actual data and more than one name can be associated with the same data. Although called by different names, any changes made will affect the actual data, regardless of how the file is called at a later time. Hard links can only refer to data that exists on the same file system.

**Symbolic link**  It is a special kind of file that points to another file in the file system and it is more like a shortcut in Windows, an alias in the Mac OS operating system, or a shadow in the OS/2 operating system. This difference gives symbolic links certain qualities that hard links do not have. For example, a symbolic link can work between different file systems.

Although links were released as additional features based on directory structure of file system to enhance the flexibility, they still have drawbacks and limitations.
• From the point of user behaviors, people make relatively little use of these links for simultaneously classifying documents into multiple categories [14] and on the other hand, file system itself doesn’t seem to encourage simultaneous classification. Many users place their files into a single directory because they are not willing to expend the effort to classify files. Of the fraction that are willing, there is yet a smaller fraction who would be willing to save their files in one place, and then separately create shortcuts, aliases or symbolic links in the other directories [36].

• Both of the links are directed links, which can only point from the name or link file to the target file but cannot point back. For example, in Fig 2.1 (a) (B, C are links to A), it’s easy to hop from B, C to A. However there is no directly way to jump back.

• They have limitations in the awareness among links and the targets. Although there are commands to know the inode number of a certain file for hard links and to get target file for symbolic links, the target files or directories are not aware of the definite sources of the links. For example, in Fig 2.1 (a), A is not aware of the existence of B, C and B, C are not aware of each other, either.

In addition to the drawbacks of the links, file system itself also has another major problem, which may also affect the efficiency and performance of the searching. Most of the file systems apply tree traversal to approach a target file; that is, in order to get a file, the system has to begin from the root and recursively traverse until get that target. That takes a lot of redundant searching time, which may affect the performance. Therefore index, which applies B-tree search solution is an ideal alternative to overcome that problem. Systems such as MSN desktop search and Google desktop search are all applying supplementary stored file to index the file on the disk and then locate targets by B-tree searching.
2.1.3 Directory-based search engines

Directory-based search engines, such as Yahoo Directory and Google Directory, is another directory-based application. Yahoo Directory is maintained by limited number of paid editors while Google Directory is based on ODP (open directory project) [30], which is maintained by a large number of volunteered editors on the internet. Both of them are trying to collect and organize the web pages online manually in a proper way, so that users can easily find the desired information within a search engine. Both of them apply the hierarchical tree structure to organize the information. They have couple of top directories, such as Education, Computers and etc., which represent the rough domains within the structure, and refined subdirectories under beneath representing the sub-domains under those rough domains. The leaves of the structure are the corresponding web page information, including the URL, description and etc. Since they organize the web pages in the directory structure, they have corresponding restrictions, such as every directory can only have one parent directory and every web page can only be placed under one certain directory.
In order to make the directory structure more flexible, directory-based search engines also introduced supplementary functions such as alias links and related links.

Alias link is more like the symbolic link in UNIX file system. The differences between the alias link and symbolic link are:

- Alias link introduces a symmetric link, which is called related link. That makes the link a bi-directed link. For example, in Fig 2.1 (b), when B throws an alias link to A, a corresponding related link is established and point from A to B.

- Due to the introduction of related links, directories become aware of the alias links pointed to them. For example, in Fig 2.1 (b), A is noted of the alias links from B, C and on the other hand, B, C could know each other in an indirect way by hoping to A to check A’s related link list.

However this directory structure still has its own limitations, since the alias links just point to targets but not keep original location or path of where the alias link comes from. For example, in Fig 2.1 (b), D is the subdirectory of A and D is not aware of A’s virtual parent directories B and C. That is because in directory structure every directory only have one path from the root and no matter how many alias links are pointed to it, it keeps that only path. But sometimes, it’s useful to get all the paths information of a certain directory. If someone approaches D from E, he may have no chance to get notice of the information within the directories along the path from B or C, which may interest him and also if someone tries to find a document under B and the target document is actually classified under D, there’s no way for him to hit the target by keyword search under B within a directory-based search engine.

2.1.4 Online posting systems

Online posting systems have been evolved from original Bulletin Broadcast System (BBS), news groups to some up-to-date forms such as forums and web logs (blogs). The aim of
these kinds of posting systems is to provide a place online for users to discuss all kinds of topics or show their personal thoughts or point of views. Since the topics may concern every aspect of people's life, they are divided into difference domains, called boards, and within each domain, there are further refined sub-domains, called sub-boards, under beneath. That also forms a hierarchical tree structure, in which every board can only have one parent board and every posting can only be posted under one certain board. For example, in the Java Developer Forum, a user may have a question which concerns both JDBC and JSP. However, due to the restrictions of the directory structure, his posting can only be placed under one directory at a time. What he can do is to post the question twice under different directories and that in fact is two independent operations for a single question. Obviously, that causes inconvenience for users who want to post questions. On the other hand, if a user wants to search for a certain solution, which spans two or more different topics, he may also miss the target if the corresponding solution is not placed under the board he is searching but another related one. Blog is another new form of posting and more and more attention has been concentrated on it. Blog search has become one of the hottest topics within search domain. Some of the work in blog searching is related to the RSS search [38] while others are trying to organize the blogs online into well categorized directories. Blogcatalog [5] is one of the forerunners to organize the blogs online. It also applies a directory structure to manage the blogs online and that leads to the same problems caused by the natures of the directory structure we mentioned above.

2.1.5 Other directory-based applications on desktop

In addition to the application discussed above, we also have other desktop directory-based applications, which concern our daily work on computers. One of them is the bookmark management console in browsers. Bookmark is one of the nice features from the browser and it makes users easy to approach the websites they are interested in. However with the
rapid growth of the web sites online, the number of the bookmarked web sites within the browser increases quickly. How to organize, manage, maintain, and search for bookmarks in browsers also becomes a challenge to ordinary users. Browsers are keeping updating their bookmark management consoles to meet the requirement. For example, Firefox provides friendly and nice interface for users to manage their bookmarks. However, its bookmark management is also based on the directory structure, where the directories are the non-leaf-nodes and the leaf-nodes are the web site information. Another most frequently used application is the email management system, such as Microsoft’s Outlook Express, in which users can organize the emails in directories. Due to the limitations and drawbacks of directory structure we discussed above, these applications also have the problems caused by the natures of the structure.

2.1.6 Summary

From the description above we can see that directory structure is prevalent within the computerized world and the limitations and drawbacks of the structure do hinder the flexibility and efficiency of the corresponding applications. It is necessary to improve the widespread directory structure.

2.2 Context-based searches

In order to handle tremendous volume of data online, more and more new enhanced technologies have been proposed to improve the efficiency and accuracy of information retrieval. Context-based is not only one of the most frequently used terms, but also one of the most ambiguous concepts in this domain. It can be referred from the text body processing technologies, such as IntelliZap [17] and Context-Based Similar Words Detection technology [21], the personalization technologies, such as PRISM [11] to the category based technologies such as Context-based Google [1].
2.2.1 Text-body-processing-related context-based search

This is one of the most natural methods to process the context-based search within the text body. Keyword-searching has been one of the most important and widespread ways of web searching ever since the inception of internet. However due to the tremendous volume of data on the web, traditional keyword-searching methods cannot avoid returning results full of fuzzy information. Therefore new processing improvements have to be done to get more accurate searching results. Text-body-processing-related context-based search starts from the original point of the keyword-searching, i.e. search within the text body. The difference is that new improvements are added during the searching procedure to attain more satisfactory results.

IntelliZap [17] is one of the interesting methods among the new text-body-processing-related context-based search solutions. It refers the context-base as the words around the searching keywords. It is separated from the traditional keyword-based search by submitting the keyword when reading a certain document. The keyword selected by users is wrapped in the phrases, sentences or even the paragraphs within the document and then processed by its own semantic networks and relevancy ranking mechanism. The results are much more focused on the same domain as the source documents. For example, if a user is reading the article about the tourism information on Java Island, when he or she submits the searching query within the article, the context information, including the phrases, sentences or even paragraphs around the keyword of Java, is sent to the system. Then by some semantic and ranking processes, the results returned are mostly about the information on Java Island, rather than Java programming or Java coffee. Hisham Al-Mubaid et al. [21] proposed a method by introducing a supplement vector structure to represent the words having the most frequent occurrence around the target keyword. By coupling the vectors and the returned results, the less relevant results are filtered.

There are many other similar approaches have been proposed in this aspect and most
of them are concentrated not only the target keyword but also the information around the keyword within the documents. That is the main context-based idea of the research in this aspect.

2.2.2 Personalization-related context-based search

Most of the web search solutions are user-independent searches. They do not refer to the individual preferences during the search. For example, a computer scientist and an astronomer will get the same search results when submitting a keyword of Eclipse to most of the web search systems. Actually what concern them are totally different things while a computer scientist is probably interested in the IBM’s open source project of Eclipse or one kind of logical programming language and an astronomer cares more about the astronomical phenomena of Eclipse. Therefore personalized search has caught more and more attention because it offers more relevant search result according to individuals separately. Quite a lot of researches have concentrated on this aspect and most work so far gets the personalized context-based search by keeping track of users previous queries or documents viewed [17].

PRISM [11] applies Watson [9] to extract personalized context-based information for more accurate search. It detects users’ daily interactions within the system, such as word processing, user actions and explicit searching requests, and saves them into the knowledge base as users’ profiles. When a user performs the web search from PRISM, his or her individual profile is drawn from the knowledge base and system returns the results much more relevant both to the content and individual interests by coupling the raw search results and personalized context information.
2.2.3 Category-related context-based search

Web directory is now widely used for organizing huge number of web pages online. The di­rectory structure and content have become more and more complicated and comprehensive since it came into being. No matter how is the structure maintained, by paid editors, such as Yahoo Directory or by many volunteers, such as ODP (Open Directory Project) [30], the refined hierarchy structure does well represent the context information of the doc­uments classified under it. Therefore many researches have been done in this aspect such as Context-based Google [1]. Since our research concern the aspect of category, we care more about the category-related context-based search. The detailed explanation is given in the later chapter.
Chapter 3

Related works

Based on our research, the widespread directory structure does hinder the flexibility and efficiency of organizing and searching for information. In this chapter, we provide a critical overview of the related works from both the academic community and the industry and point out the novel features of our new multiple-categorization hierarchical structure.

3.1 Utilizing the semantic meaning of hierarchy structure for web searching

With the development of the directory structure, it has been expanded from original prototype of simple tree structure to more complicated hierarchy structure. People are becoming to notice that the hierarchy information does represent the semantic meaning and can outline the contents, including subdirectories and documents stored under a certain directory. Because of the resourceful meaning from the directory hierarchy, computer scientists are trying to combine it with ONTOLOGY, the novel concept from AI community. An ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence. Within AI community, the
Existence means something can be represented. By combining the concept of ONTOLOGY and directory hierarchy, computer scientists proposed the method of representing the information by the directory hierarchy to which the information belongs. Then the representation can be processed by some logic programming system and do the retrieval or classification jobs.

Yannis Labrou et al. [24] proposed an ONTOLOGY-using of Yahoo directory to describe the documents under each directory. Since Yahoo directory is one of the most comprehensive directory structures and it has refined directory hierarchy and detailed description for both Yahoo directories and non-Yahoo URL, they take advantages of these and build corpus for each directory within Yahoo by directory name, directory hierarchy and non-Yahoo things, including entry title, entry description and entry content. Then the corpus is treated as the ONTOLOGY representation of the directory. When a document comes, the system can evaluate the weight of the document by calculating the differences between the document and the corpus with Telltale [34] and then it can be classified according to the weight values.

The semantic meaning of category hierarchy is not only used for automatical information retrieval and classification, but also used for personalized searching. Fang Liu et al. [26] proposed a theory to map users’ queries into categories to attain their search intentions. They capture users’ searching intention by record both users’ actions on documents and the category selected before submitting a query. In addition to establishing user profiles, they also build general user profiles according to both the individual profiles and the category hierarchy information from ODP structure dump [31]. With these user profiles, their system can be noted about the users’ search intentions.

Although automatic classification and retrieval by semantic method is an ultimate goal, researches in this domain are still far from high accuracy and flexibility in all circumstances. The relative high performance and accuracy of the researches so far have been
based on the control of sample corpus, parameters, specific domain or the selection of automatic classifier algorithm. Therefore they may get ideal results with respect to a very restrict and specific scenario. However no work so far has been declared and proven as generic usage. Human’s interference is still the mainstream of information classification and index. Projects such as Yahoo Directory, Open Directory Project, Wikis and etc. all suggest to maintain the information manually. Different from automatic information classification and retrieval by utilizing semantic meaning of category structure, our work is to provide a user interface based on category structure for users to manually manipulate and retrieve information. We encourage arrange information manually by users, who have more accurate and flexible understanding of information than machines. We provide such a category structure that information within that structure are continuously well maintained by public users.

3.2 Faceted classification and library systems

Faceted classification allows one to model applications with complex classification hierarchies using orthogonal dimensions [37]. This means every entity can be classified to multiple hierarchy structures and can be retrieved according to any of the structure independently. This has been widely used in the library systems, since a book or digital record within the library system can belong to more than one domain. Library Of Congress Online Catalog, is one of the largest libraries in the world. It applies multiple classification tools to manage over 128 million items [8]. For instance, Library of Congress Subject Headings (LCSH) is applied to every item within a library’s collection, and facilitates a user’s access to items in the catalogue that pertain to similar subject matter. Every item can be categorized to multiple subjects. Despite LCSH’s wide-ranging and comprehensive scope, it is not an ideal and general structure for common usage. Dewey Decimal Classification (DDC) is another classification tool of library. It covers the topics by a three-level hierarchy
structure, in which there are 10 main classes, 100 divisions and 1000 sections. Multiple categorization is attained by a series of three digit numbers delimited by decimals. These classification alternatives all emphasize on hierarchy structure transparent, i.e. users are not aware of the structure.

S. Geffner et al. [18] proposed a nice method to handle the category structure within the large digital library. They emphasize on the importance of numeric range, which narrows the scope of searching. They utilized data cube [20] to handle the numeric analysis. Numeric information, such as date, accompanied with documents in digital library system can be aggregated and analyzed by multi-dimensional databases such as OLAP and the result is a summary of the numeric statistics of the result distributions, such as the date range. By selecting the numeric range, users can narrow the search result and pick up the target in a relatively small scope. Illumined by this nice method, they expand this to the non-numeric domain, such as the category hierarchy. The original aim of the hierarchy structure is the same as the numeric range, which is to narrow the searching space. By digging deeper in the category structure, users can narrow the possible candidate results. Based on this similarity, they propose a novel method to evaluate numbers to categories, whose numeric ranges enclose the numeric ranges of all their subcategories. By doing so, they can utilize the data cube to aggregate and analyze the numeric ranges, which represent different categories. Due to the multi-dimensional characteristics of the data cube, they can apply multiple category structures, which lean to different aspects, to the searching results. [40] proposed a multiple-dimensional method to classify the text instead of applying flat and hierarchical structure, and proved that it was likely to attain higher automatic classification accuracy. Dennis Quan et al. also implemented a multiple-categorization interface embedded in browser and they prove that multiple-categorization are more flexible and natural to organize information from the aspect of human behaviors [36].
Faceted classification within library systems is a nice way to classify the books. However they pursue the hierarchy-structure-transparency, i.e. users are not aware of the subject hierarchy structure under beneath. This is specific for library usage and not portable for general application. In addition to that, the hierarchy structures, such as Library of Congress Subject Headings (LCSH), seem rigid and non-expansive while Dewey Decimal Classification (DDC) lacks meaningful representations with a series of decimal numbers. [18] and [40] emphasize on the internal structure and ignore the importance of user behaviors and user interfaces. [36], on the other hands, puts more efforts on the aspect of human behaviors and releases interfaces, which encourage people to multiply classify information. However it still has its structure semi-transparent. From its survey around the professional users, it seems that users are willing to get more information on the hierarchy structure, including listing all the parent level categories during the search. In our work, we propose the category structure, which can handle the multiple-dimensional classification in an extensible and portable way. Meanwhile our structure is likely to provide resourceful information about the structure to end users for accurate and relevant search.

3.3 Category-based file management system

In addition to the related works above, which concern more about the representation or the semantic meaning within the category hierarchy, [44] focuses on how to apply the category structure to an existing widespread application and proposes a category-based file management system, called CASH, based on the UNIX file system. [44] points out the similar idea to our research, which regards category structure as a more natural and flexible way to organize and search for information. CASH is established by extending TCSH, which is one of the frequently used shell on UNIX system, so that users can manipulate and search for the files within the Unix file system in the way, which is similar to using the commands under Unix file systems. Although CASH is an inspiration to our research, it
still has many limitations and drawbacks, which are overcome in our work.

- **CASH is platform-dependent**: CASH is built based on extending TCSH, which is a shell on UNIX and then CASH can only be used on a Unix-based system.

- Although CASH can load the existing UNIX file system into category structure, what it concerns is just the directories and files and it does not check the existing links within the structure. However the existing links within the file system is also very important to illustrate the hierarchy relationship among the directories and files. In addition to the file systems, other applications, such as directory-based search engines, also have links, such as alias links and related links, which represent the supplementary hierarchy relationships besides the directory structure. Therefore mapping those existing links into our category structure is also necessary.

- **CASH is based on command line**, which is too boring and complicated for unprofessional users. Most of the ordinary users prefer graphic interfaces, such as web interface.

- **Categories within CASH system does not apply all the possible paths information from the root and to our research**, we think it’s much more presentive and flexible if we can be noted of all the possible paths from root to a certain category, and consequently we can perform not only the up-to-down searching method, but also the bottom-up searching solutions.

- **Most of CASH’s work is concentrated on category manipulations such as create, edit or remove and limited in searching ability.** The only search command `lscat` may work well if only we search within limited number of information and give relatively complete keyword. Otherwise, the screens’ of results will cause much inconvenience.

- Although CASH applies relational database to store and manipulate the category structure, the database schema is not presentive and effective to process in a DBMS.
3.4 Knowledge Management

Knowledge Management (KM) is one of the most exciting management subjects of today and it concerns almost everybody's daily life, including individual information exchange, e-commerce, e-government and etc. As a formal theory and course of study, KM has developed in response to advances in technology - the scope and speed by which knowledge is accessed and exchanged. Individuals and companies are seeking better ways to harness this knowledge potential, so KM attempts to provide the answers via real-world explanations and best practice methodologies. Therefore different definitions are given to Knowledge Management with respect to different aspects. For example, from the point of e-commerce, Knowledge management is an effort by organizations to manage some or all of the knowledge within them as a resource, much as they manage real estate, inventory, and human resources. It involves the aspects of: explicitly recording the tacit knowledge within an organization, cataloging and storing the knowledge, transforming knowledge for use in other contexts and disseminating knowledge [33].

What is knowledge? Nonaka and Takeuchi apply the concepts of data, information, tacit knowledge and explicit knowledge to describe several types that are relevant to an organization. Tacit knowledge is knowing that has been internalized, and is difficult to share or express. Explicit knowledge has been externalized, and is easier to exchange. Nonaka/Takeuchi conclude that all knowledge creation begins with the individual [32]. Don Mezei [13] describes the difference between the levels of the knowledge: data is the facts and figures out of context, information is data that has value to someone in the context of their work or life and knowledge is information that people make use of.

Why is Knowledge Management so valuable? One of the most important factors, is the value of sharing knowledge, sharing explicit knowledge. Today KM-software, like Outlook and Lotus Notes, is very much part of the every day work life of any employee in the IT or telecommunications industry. The main benefits of such tools are that they support
organizations in expressing, sharing and retaining information and knowledge. Other practical Knowledge Management tools, such as Wikis, Google Answers, iAsk.com are more closer to computer users online. The common point of these tools is that they offer a place to organize knowledge and information related to all the things in which people might be interested in an open society. With the rapid growth of AI community, some other scientists are proposing the next generation Knowledge Management, which apply ideas such as meta-data, ontology to attain machine-processable semantic management [23].

To some extend, Knowledge Management is a rather bigger domain, which encloses our research. It concerns not only how to record, store and classify information, but also how to transform and apply the information as knowledge to handle practical problems. In another word, our research work is trying to address some aspects within Knowledge Management. We are trying to propose a flexible and efficient structure and interface for explicit knowledge management within knowledge management systems. There are two typical knowledge management tools, Wikis and Content Management Systems. Wikis [10] organizes knowledge and information related to all areas in which people may be interested in an open society. Content Management Systems [6, 2] are computer software systems for organizing and facilitating collaborative creation of documents and other content.

In general, Wiki [10, 4, 29] is a simplification of the process of creating HTML pages combined with a system that records each individual change that occurs over time, so that at any time, a page can be reverted to any of its previous states. It provides a nice space for online users to share their knowledge. Wikis are a true hypertext medium, with non-linear navigational structures [10]. Each page typically contains a large number of links to other pages. Theses links could be words within the text body or the related reference lists. Therefore the original Wikis is a network of web pages linked by hyperlinks within pages. However, most of the Wiki systems only provide a flexible method and mechanism for users to add and edit knowledge, and lack well-formed category structure to organize knowledge and information. For example, Wiktionary organizes the content by alphabets,
which lacks the semantic meaning of the structure. Some other bigger Wikis are applying hierarchy structures to arrange tons of web pages. For example, Wikipedia provides a nice category structure, which also allows categories and web pages be multiply categorized. However it still lacks user-behavior-preferred interfaces [36] and it has limited searching options. For example, although every category or web pages can be multiply categorized, it only lists one parent category. This does not fulfil users’ potential request for more related parent information as mentioned in [36]. In addition to that, Wikipedia only provides category digging deeper search and keyword search. They are only efficient to users, who have complete ideas or keywords of the target. If a user is not familiar with the target (digging deeper operation is not efficient) or the number of result is huge due to the incomplete keywords (keyword search is not efficient), Wikipedia performs inaccurate and inefficient searching. Our work enhances the interfaces, which provide more relevant parent information during the digging deeper operations. Meanwhile, we also provide an additional bottom-up search solution, which can wrap the keyword searching result into subset of the category structure and offers more accurate and efficient search.

Documentum [15], FileNet [41] and Hummingbird [27] are three famous enterprise level content management systems. They are very strong tools to manage all kinds of medias or formats of documents. In addition to the ordinary information or content management, the main purpose of these systems is to provide business analysis based on contents and coordinate the cooperation within the business. They concern many business concepts such as team collaboration, customer relationship, supply chain and etc. One of their key points is also how to organize documents into categories of interest and simplify their retrieval in later using. However these systems are still specific for heavy weight enterprise usage and lean to business processing. Although they do offer categories to organize information, the category structure is, to some extend, semi-transparent to users. Information is managed automatically according to business rules and semantically processed. As we mentioned in the previous section, our work is to encourage manual information
management within a flexible category structure, which is explicit to users and maintained by public users. In addition to that, different from domain specific usage of the Content Management Systems above, our systems can be used not only over all kinds of platforms, but also from lightweight personal usages to large scale public online communities.

3.5 Email management systems

Email is one of the most important usages in computerized world and it concerns about everybody’s daily life. With the time goes by, bunches of emails are piled up and how to manage the emails becomes a challenge to users. Systems such as Microsoft’s Outlook Express does simplify most of the manual job by organizing the email into directories (folders) and users can search the emails by directories (folders). However directory structure can no longer meet the processing nature of the emails, such as a certain email can belong to multiple topics or groups. These drive computer scientists to figure out a more flexible and natural structure to organize and search for emails.

Bloomba [19] is not just an e-mail client, it’s specifically designed for people who are overwhelmed by the amount of e-mail they receive. Bloomba has many nice and strong features, which raise the efficiency of people’s daily work with emails. In addition to the functions such as strong spam ability, smart group, portable datastore and etc., we concern more about how it organizes and searches for emails. It introduces the idea of category and every email can be classified to multiple categories and displayed with different colors. It also automatically groups emails according to subjects. However it only applies a flat category structure, which only has one level categories. Although emails can be classified to more than one categories, it still take efforts to retrieve target email when the number of emails or categories is very huge. A hierarchy structure is necessary. We provide such a hierarchy structure, which allows emails to be multiply categorized. That helps to refine email classification and simplify email retrieval.
Yahoo Mail [25] is also a famous mail system, which offers its own specific email search. It applies a similar method to the one mentioned in [18]. Since email has its own specific properties such as sender, attachment, data and etc., Yahoo Mail takes these as multi-dimensional structures. For example, if I input *thesis* into my Yahoo Mail search, i will get 106 results, which are categorized according to different properties, such as senders, dates or attachments. Since I am going to find the one with my pdf copy of thesis, it’s easy for me to pickup the target email from the 106 emails by selecting the category of *attachments*. This gives us a nice idea to wrap the searching result into corresponding category so that users can pick up the target much easier than search in the result list. However it is strict to the flat structures of the dimensions. Although emails are multiply categorized to more than one dimensions according to properties of emails, it may still take efforts to retrieve emails within every individual dimension. For example, if a user has more than 1000 contact information in his contact list, it may take time to locate the target emails by senders. Based on our structure, users can freely apply multiple-categorization based on the hierarchy category structure and perform more efficient and flexible huge volume email management.
Chapter 4

Multiple-categorization hierarchical structure

Although the problems of the directory structure are widely recognized and much work has been done to improve it, most of the work so far has been concentrated on their own specific usages. Our work is to establish a portable, extensible category structure, which is natural and efficient to manipulate. In this chapter we formally define the notions involved and discuss the important features of our category structure.

4.1 Category

What is a category? Generally speaking, category is referred to any of several fundamental and distinct classes to which entities or concepts belong or a division within a system of classification. A category illuminates a relationship between the subjects and objects of knowledge. A category has a user-defined name to describe the commonalities of all files or information that belong to it. A category name can be any noun or noun phrase, but should be meaningful to its users. It helps users to organize their files and reminds them of what their files are about. A category name usually shows one common aspect of a group
of files or information. Sometimes, it also can be an arbitrary value that reminds users of one of features of files, such as file format, but does not have to be a word in corresponding files.

No matter what kind of structures, flat or hierarchy, directory or category, they all intend to organize the documents on local or remote machines. Therefore documents are the basic elements within a category structure. Documents could be stored on local machine such as those in the local file systems. They could also be placed on the remote servers, such as search engines, which organize, index and retrieve documents on servers throughout the whole internet.

In order to handle thousands of millions of documents, we try to divide the documents into groups or relevant topics and organize documents by manipulating those groups or topics. We call those groups or topics categories. Due to the tremendous volume of documents, even if they are grouped into categories, which concern quite a bunch of aspects, the number of categories is also becoming huge and beyond what users can efficiently handle. Therefore a hierarchy structure based on those categories becomes necessary. Categories are further grouped into different levels of domains from rough to refined and form a hierarchy structure with a root. From the root we could get top level domains, which are the rough domains of the documents in the system. By digging deeper into the lower level categories, domains are narrowed and refined. The refined categories filter the un-relevant documents and lead to the target ones.

As we mentioned above, category names play an important role in the structure. They could be the brief description of the documents under beneath, such as keywords of the documents. For example, category *Computers* represents the group of documents, which are related to computers. They could also be arbitrary values, which remind users of features of files. For example, we could have categories such as *PDF, DOC, JPEG* and etc. to classify documents according to their formats and later on we can find our target documents not only by topic, but also by format. A category or document may have more than
one names with respect to its different parent categories. Therefore we can not identify the category or document entity in our structure by names. For example, we have a category, which enclose the information or documents of video games. We could categorize this category to both categories of *Computers* and *Games*. Since under *Computers* we only need to name the category as *Games* and establish a path of *Computers/*Games*, which could represent the meaning of video games under category of *Computers*. On the other hand, under the category of Games, we could all have all kinds of games, including ball games, sports games and etc. and in this case we need to explicitly name it as *Video Games*. In addition to multiple names of categories or documents, categories or documents may also share the same names within the structure. For example, we could have category named as *JDBC* under every database categories, such as *DB2*, *ORACLE*, *MYSQL* and etc. or categories named as *Faculty* maybe placed under every universities. Therefore we can not identify the actual meaning of the categories or documents merely according to their names. We should refer to their context information (parent categories information) to derive the actual meaning of the target categories or documents. For example, *Computers/*Software/*Databases/*Oracle/*JDBC* and *Computers/*Software/*Databases/*DB2/*JDBC* represent different categories, though they have the same category names. Based on the two points stated above, we can see that names can not be used to identify the category or document entities within our structure and we’d rather taking names as the mappings of categories or documents in their parent categories.

Different from directory structure, a document or category within a category structure should be able to be categorized to more than one categories. For example, an academic publication could be categorized to multiple categories, such as topic, author, date and etc. Each category may represent one dimension information of the publication. Therefore the publication could be retrieved by any of the dimensions or the combinations of those dimensions according to users’ preferences. Due to multiple-categorization in category structure, we could get multiple paths for a certain category or document. The path information also
represents the semantic meaning of the target.

Hierarchy structure on categories represents different levels of domains. It could concern both the relationship of inheritance and composition. For example, category Programming languages could have sub categories such as Java, C, PHP and etc., which are programming languages (inheritance). We could also have category Carleton University, under which there are categories such as Students, Faculty, Courses and etc., which compose the university. Therefore higher level categories generally have larger or rougher scope of a domain. However we could also have relationships between the domains other than hierarchy relationships, such as two related topics. We should have special links to establish relationships between two related categories, which have neither inheritance nor composition relationship. For example, Computers/Computer Science/Academic Departments is related to the domain of education and we should establish a link between these two domains. When approaching Computers/Computer Science/Academic Departments, we could be noted of its related information under the category of education and get further information about schools.

In the following we formalize the general concepts of category structure with respect to our work and provide more detailed definitions and features description of our category structure.

4.2 Definitions

Here are the corresponding definitions of our category structure:

**Definition 4.1 [Category]**: A category is an object in the category structure that can have a group of a group of child-categories and documents. It is represented as a node in a directed graph. A category has the following properties: the number of child-categories and the number of documents in it, its level in the category structure, all the possible paths
Figure 4.1: Category Structure
from the root and last modified time.

Definition 4.2 [Document]: A document is an information unit and is typically stored in a file system within a local or remote computer. It is represented as a node in a directed graph. A document has the following properties: its location, size and last modified time.

For example, in Fig 4.1, circle-nodes represent categories and rectangle-nodes represent documents, which are files in a computer.

Definition 4.3 [Category/Document name]: A category or document name is a classification of the category or document in the parent category. This name of the category or document is only meaningful and relevant with respect to its parent category. It is represented as a named directed edge in a directed graph. Let $c_1, c_2$ be two categories. If there is a category name $n$ and $n$ is the name of $c_2$ with respect to $c_1$, then $c_1$ is a parent category of $c_2$ while $c_2$ is a child category of $c_1$.

For example, in Fig 4.1, category 22 has two category names, called Eclipse with respect to category 11 and Eclipse Project with respect to category 14. A child category must have a level lower than its parent category (see Definition 4.5).

In our category structure, we disallow cycle by using the level function to be introduced in Definition 4.5. A parent category has a smaller level value than a child category. However, we may still need to classify a higher level ancestor category to a lower level descent category to be flexible in practice. For this reason, we introduce another kind of classification.

Definition 4.4 [Virtual category name]: A virtual category name is a special classification of the category in the parent category. It is represented as a named directed edge.
between category-nodes in a directed graph. Let \( c_1, c_2 \) be two categories. If there is a virtual category name from \( c_1 \) to \( c_2 \), then \( c_1 \) is a virtual parent category of \( c_1 \) while \( c_2 \) is a virtual child category of \( c_1 \).

For example, in Fig 4.1, there’s a virtual category name \textit{Computer Science}, which classifies category 6 to category 18. Unlike category/document names, a virtual child category can have a level higher than its virtual parent category.

\textbf{Definition 4.5 [Multiple-categorization hierarchical structure]:} A multiple-categorization hierarchical structure is a rooted directed-acyclic-graph

\[
S = (C \cup D, N, V, r, l, p, in, out)
\]

where \( C \) is a set of categories, which are recognized by category id and \( D \) is a set of documents, which are recognized by document id,

\( N \) is a set of named edges for categories and documents,

\( V \) is a set of virtual named edges for categories,

\( r \in C \) is the unique root of the graph,

\( l \) is a function that maps a category to a number which is the level of the category in the graph,

\( p \) is a path function that maps the category to its paths from the root,

\( in \) is a function that gives the number of incoming edges of a node,

\( out \) is a function that gives the number of outgoing edges of a node,

such that the following hold

1. \( l(r) = 0 \); that is, the highest level has the least level value.

2. For each \( c \in C \) other than \( r \), \( in(c) \geq 1 \) and \( out(c) \geq 0 \); that is, a category other than root must have a parent category and can have 0 or more child categories and documents.

3. For each \( d \in D \), \( in(d) \geq 1 \) and \( out(d) = 0 \); that is, a document must have a parent category.
(4) Let \( c, c_1, c_2, \ldots, c_n \in C \) such that for each pair \( c_i \) and \( c \), there is a directed edge \( n_i \in N \) from \( c_i \) to \( c \); that is, \( c \) is a child category of \( c_i \), and then \( l(c) > \max(l(c_1), l(c_2), \ldots, l(c_n)) \) and \( p(c_i) \subseteq p(c) \) for \( 1 \leq i \leq n \).

(5) Let \( c, c_1, c_2, \ldots, c_n \in C \) such that for each pair \( c_i \) and \( c \), there is a directed edge \( v_i \in V \) from \( c_i \) to \( c \); that is, \( c \) is a virtual child-category of \( c_i \), and then \( l(c) \leq \max(l(c_1), l(c_2), \ldots, l(c_n)) \) and \( p(c_i) \not\subseteq p(c) \) for \( 1 \leq i \leq n \).

For example, Fig 4.1 shows a sample multiple-categorization hierarchical structure. Category \( 0 \) is the root of the structure and its level value is 0. The circle nodes in the figure represent categories, which have at least one parent category and have 0 or more child categories and documents. The rectangle nodes represent documents, which also have at least one parent category. Category \( 22 \) is the child category of both category \( 11 \) and category \( 14 \), and it's in level 4, which is lower than its parent categories' (category \( 11 \) and category \( 14 \) are in level 3). The path of category \( 22 \) includes both its parent categories' paths (Top/Computers/Companies/IBM/Eclipse Project and Top/Computers/Programming/Java/Eclipse). Category \( 6 \) is the virtual child category of category \( 18 \) and category \( 6 \) is in the level higher than category \( 18 \) so that the path of category \( 6 \) dose not include that of category \( 18 \) (path of category \( 6 \) is Top/Computers/Computer Science).

From the definitions above, we can see that multiple-categorization hierarchical structure is a rooted graph structure to organize and manage over documents. Every category and document within that structure can be categorized to more than one categories so that information can be classified and retrieved in a flexible and efficient way since users can handle the target documents by different aspects of interests.
4.3 Properties of the structure

Here are the featured properties of our category structure.

-Multiple categorization
A category or a document can be categorized to multiple categories. For example, in Fig 4.1, categories 19,20,21,22 and documents 7,9 are all categorized in multiple categories. This new feature loosens the restriction of the directory structure and makes it more natural and flexible to organize and classify information.

-Multiple paths
Due to the multiple categorization feature, every category, except for the root, may have multiple paths from the root category. This is one of the biggest differences between the novel category structure and the directory structure, where every directory has only one path from the root. The reason for emphasizing on the multiple paths information is that it provides users much more related information during the searching. For example, in Fig 4.1, if a user approaches the category 22, he or she will get two paths from the root, Top → Computers → Companies → IBM → Eclipse Project and Top → Computers → Programming → Java → Eclipse. This multiple paths information provides more related information of a certain category and it's easier and more flexible for users to retrieve desired information than one path information provided by directory structure.

-Categories or documents with same names
In some directory structure applications, such as file systems, same names under a certain directory is not allowed. But this is allowed within our novel category structure. Sometimes documents or categories with the same name may have different contents. For example, in Fig 4.1, categories 19, 20, 21 are child categories of category 11 and they all have the same
category name **JDBC**, though the information under them are different (JDBC for DB2, ORACLE, MYSQL separately).

**-Same category or document with different names**

This is also a nice feature for naturally organizing and searching for information. Category name or document name in our category structure is a mapping in their parent categories. It is used for users to recognize the target information. However when describing a certain document or category, different people may have their own preferences. For example, for the academic papers, someone refers it as Papers while others may name it as Publications. Similar situations are all over the computerized world, such as Database can also marked as DB, Computer Science can be CS and etc. Therefore, in our category structure, we allow different category or document names for a certain category or document. For example, in Fig 4.1 category **22** has two different category names, **Eclipse** and **Eclipse Project** and document **7** also has two names, **SCS CU** and **School of Computer Science**.

**-Child category and virtual child category**

There are two kinds of child categories in our category structure, child category and virtual child category. The main difference between them is that child category is from the category classification while virtual child category is from the virtual category classification. Another important difference between these two child categories is that child category is classified under its parent category so that it holds the path information of its parent category, i.e. path of parent category is the subset of path of the child category. On the other hand, virtual child category is just a virtual mapping from from its virtual parent category and does not hold the original path information of its virtual parent category. The reason for using virtual category classification is because that it loosens the category classification restriction (level-dependent, i.e. categories can only be classified to higher level categories), which is to avoid cycle within the structure, meanwhile it does not violate the
acyclic-directed-graph principles. What's the difference between the virtual child category and hard link in the UNIX file system? Although we can approach the target file or directory via any of its links, we could not get all the links via a target file. However our virtual child category are noted of all of its virtual parent categories. This makes our structure can apply not only an up-to-down search solution, but also a bottom-up approach.

After describing the basic concepts and properties of the category structure, we explain how to model the abstract concepts in the coming chapter.
Chapter 5

Category structure implementation

From the descriptions in the previous chapters, we can see that we have described a category structure, in which there are some new features beyond the existing structures and systems. Although CASH [44] has proposed the same idea as our work, it is poorly designed and modeled, especially in database schemas. In addition to that, CASH work mainly extends the TCSH on UNIX file systems and is specific for file management in the file systems, while we are going to propose a more generic structure and interface for not only file management, but also information management online. Therefore we need to build up a more platform-independent system with more friendly user interfaces, since most users are likely to prefer graphic user interfaces than command line based interfaces. So we implement our system based on web technologies and redesign the models and schemas in CASH system.

5.1 Overview of system architecture

We have implemented our experimental system in Struts framework, which is a Model 2 framework, and manipulate our category structure in MySQL DBMS.
MySQL DBMS  Why do we apply database to manipulate and store the category structure? Actually we have many alternatives, such as storing the structure in a file on the disk like most of the file systems do or saving the structure in the registry table as Windows applications do. The reasons for picking up database system are:

- database management systems provide nice features such as index, which obviously improve the searching speed than tree traversal applied in most of the file systems. It should be theoretically more efficient to retrieve record by index than by tree traversal.

- most of the database management systems provide strong and nice features such as concurrency control and other operational optimizations, which also help to organize huge volume of data, such as information within a search engine, in an efficient way.

- database management systems also provide resourceful APIs and interfaces for application developments

Based on the reasons above, we apply database management system to store and manipulate our category structure.

MySQL is characterized as a free, fast, reliable open source relational database management system. It is under the GNU General Public License (GPL), which allows users to use it for free. It is faster than most of the existing relational database systems such as PostgreSQL, Access, MS SQL-SERVER and etc. It is much easier to use than giant commercial aimed database systems such as ORACLE, which needs well and heavily trained administrators. It is portable because it can be used on almost all kinds of platforms, including Windows, UNIX, Linux. In addition to the nice features of MySQL over other relational database systems above, another reason for using MySQL to store and manipulate the category structure in our system is that our system is a web interface application and based on web application framework, the Struts framework, which offers flexible and
resourceful APIs for MySQL database. By using MySQL, it not only makes our system work more efficiently, but also simplifies our implementation.

**Struts framework** When mentioning web application development, we have mainly three options: J2EE, LAMP (Linux+Apache+MySQL+PHP) and ASP.NET. ASP.NET is proposed by Microsoft and it is generally based on Microsoft platform and technologies. Therefore there are more compatible issues in ASP/ASP.NET than the other two options. LAMP, on the other hand, is supported by open source communities. Although it is expanding rapidly in market possession, it still lacks the maturity and supporting from wide range of the industry. J2EE is proposed by SUN and has been developed over platforms for many years. It is not only platform-independent but also much more mature than LAMP. It also gets wide supports from the industry. Therefore we pickup Struts framework, one of the J2EE web application frameworks, in our implementation.

Frameworks in J2EE communities have been thriving these years and quite a bunch of frameworks have been released. Although newer "light weight" MVC frameworks such as Spring and Tapestry are likely to compete with Struts, Struts is still a very well documented, mature and popular framework for building front ends to web applications. It encourages application architectures based on the Model 2 approach, a variation of the classic Model-View-Controller (MVC) design paradigm [22]. The MVC model totally separates the presentation, business logic and controller, and it enables developers concentrate on one of these aspects at a time, which enhances the code reusability and simplifies the maintenance. Fig 5.1 shows the diagram of the Struts framework, from which we can see that the model part deals with the interactions between the system and the database. The controller part is the mediator between the view and the model part. The view part is the presentation to users’ web browsers and interacts with users. Using this framework, it saves us time to implement a relatively complicated system.
Fig 5.2 shows the system architecture, from which we can see that the system is generally divided into four layers: interfaces, controllers, models and data.

**Interfaces**  They are used to directly interact with users by both getting users’ commands and displaying corresponding results and responses to users. Since there are two types of users in our system (we discuss it in detail in chapter 7), we have two separate interfaces: Admin Console and User Console.

- **User Console**: This is the interface for those who are authorized as the role of users. Users can both edit their own documents, including document uploading and classification, and search for information, including category search and document search.

- **Admin Console**: This is the interface for those who are authorized as the role of administrators. Administrators’ authorities cover those of users; that is, administrators can execute all users’ operations. In addition to that, administrators can also manipulate category structure and user account information on Admin Console.
Figure 5.2: System architecture
Controllers  They are used to derive users’ commands from interfaces, pick up the corresponding business object models to do the processing and response with the results.

- Category Manipulation Controllers: This is used to manipulate the corresponding category operations from Admin Console, including Add, Remove, Move and etc. By deriving the commands from Admin Console, the controller picks up the corresponding business object models to do the operations (we discuss it in detail in the coming sections) and responses with the operation results.

- Document Manipulation Controllers: This is used to handle the operations on documents from both Admin Console and User Console. The controller derives the commands, including document uploading, classification, remove and etc., determines corresponding models and responses with the operation results.

- User Controllers: This is used to do operations on users’ account information from Admin Console.

- Category Search Controller: When users request for category search, category controller will retrieve the target categories and response to users’ interfaces.

- Document Search Controller: When users request for keyword search for documents, document search controller retrieves the related documents by keyword and the results will be sent to the Category Coupling and Wrapping Controller to do further processing.

- Category Coupling and Wrapping Controller: It gets result documents together with their accompanied category information from Document Search Controller and aggregates them into corresponding categories so that it can wrap these categories and send the result back to users’ interfaces (we discuss the corresponding enhanced context-based search algorithm in detail in the next chapter).
Models They are used to directly handle the business logic, database access and file system access. The models are generally divided into two levels: Business Object Models, Data and File System Access Object Models.

- **Business Object Models**: They are used to describe the business logic of the individual operations. They apply interfaces of data access objects to manipulate the persistent data in the database and interfaces from File System Access Objects to handle the files on the disk. They offer general interfaces for controllers to refer.

- **Data Access Models**: They are used to handle the persistent data within the database and offer interfaces for business objects to invoke.

- **File System Access Model**: They are used to handle the files on the disk.

Data This layer is divided into two parts: database and documents on the disk. The persistent data stored in DBMS is the category structure, which organizes the documents on the disk. In our system, we apply MySQL to store the persistent data of our category structure.

After briefly describing the architecture of our system, we discuss how category structure is stored and manipulated in MySQL database system in this chapter and how the enhanced context-based search based on our category structure performs in the next chapter.

5.2 Alternative modeling methodologies

Object-oriented analysis and design is a common buzzword today and object-oriented methodology seems be able to well handle the design problems concerning every aspects of human life. There are quite a lot of object-oriented analysis methodologies such as DeMarco Structured Analysis [12], Yourdon Modern Structured Analysis [45], Shlaer and
Mellor OO Analysis [39] and etc., and object-oriented design methodologies, such as Yourdon and Constantine Structured Design [46], Wasserman OO Structured Design [3], Booch OO Design [7] and etc. The common idea behind these methodologies is that a computer program is composed of a collection of individual units, or objects, that act on each other, as opposed to a traditional view in which a program is a list of instructions to the computer. Each object is capable of receiving messages, processing data, and sending messages to other objects. Inheritance is one of the most important features of object-oriented methodology. It is a mechanism for creating subclasses, inheritance provides a way to define a (sub)class as a specialization or subtype or extension of a more general class. An object can inherit more than one parent class or super class. This seems a tight-coupled idea to our multiple-categorization hierarchical structure. Shall we apply this to model our category structure?

First of all, let’s check the original usage of inheritance. Inheritance is the "is-a" relationship rather than the "has-a" relationship. That is to say, the hierarchy relationship is not a composition relationship. However our category structure represents not only the "is-a" relationship, but also the "has-a" relationship. For example, we have the hierarchy structure such as Programming Languages/Java. In this case it is the "is-a" relationship, since Java is a kind of programming language and this can be represented in an inheritance format. In addition to that we may also have hierarchy such as Carleton U./Faculty and in this case, since faculty is not Carleton U., it is a composition relationship. Therefore inheritance is not suitable for this scenario. Although there are object-oriented patterns to attain compositions and components aggregation, they are still strict to the tree structure.

Except for object-oriented methodology, we have another alternative, the entity-relationship methodology to model our structure. ER has been around as a modeling technique since 1976 and it is generally used to database design. Since our goal is to apply database to store our category structure, ER methodology seems a suitable way to model our structure. Although compared with object-oriented method, ER method is inadequate for modeling
the behavior, it is great for modeling the structure of a system. Since our goal is to model a directed-acyclic-graph, we emphasize more on the structure and relationships among the categories rather than the behaviors of individual categories, ER analysis seems good for our analysis and design. Furthermore, ER modeling is simple and tight-coupled for mapping the design into relational DBMS, which is applied in our implementation. Although we can apply object-oriented database systems or mapping the object-oriented design into RDMBS, they are not as direct and simple as that of ER modeling and may bring additional overhead in the implementation.

Based on the analysis above, we apply Entity-Relationship methodology to model our category structure.

5.3 Map the category structure into database schema

According to the definitions we describe in the previous section, our category structure is a directed-acyclic-graph, which contains two kinds of nodes and two kinds of directed edges. The category structure is first represented using the Entity-Relationship model diagram in Fig 5.3, which has following entities and relationships:

- **Category (Entity)**: A category has a unique category identity (Cid), its level in the category structure (Level), the number of its child categories (Cnum), the number of its documents (Dnum), its owner’s identity (Uid) and the last modified time (Mtime).

- **Document (Entity)**: It is an entity used to describe document information and properties. A document has a unique document identity (Did), size, its accessibility state (Open), the number of its names (Dncount), its owner’s identity (Uid) and last modified time (Mtime).

- **Category classification (Relationship)**: It is a relationship based on the category set. It describes the parent-child relationships among the categories and the category
name is the corresponding mapping from the parent category to the child category. A category classification relationship has the parent category identity (Parentid), child category identity (Cid), category name of the child category (Cname), relationship create time (Ctime) and description.

- **Document classification (Relationship):** It is used to illustrate the classification relationship between documents and categories and the document names are the corresponding mappings from categories to documents. A document classification relationship has the parent category identity (Cid), the document identity (Did), the document name (Dname), create time (Ctime) and description.

- **Virtual category classification (Relationship):** It is another relationship based on the category set. It describes the virtual parent-child relationships established by the virtual category classification and the virtual category names are the mappings from the virtual parent categories to virtual child categories. A virtual category classification relationship has the virtual parent category identity (Vparentid), the virtual child category identity (Vcid), the virtual child category name (Vcname), create time (Ctime) and description.

- **URL (Entity):** A URL entity represents the real location of a document on the disk and it has the property of the document identity (Did), the document location on the disk (Location) and description.

- **Location (Relationship):** It is used to describe the relationship between documents and their real locations on the disk.

Therefore based on the definition and Entity-Relationship model analysis above we can derive the corresponding database tables (see Fig 5.4): Category, Document, Category classification, Document classification, Virtual category classification and URL.
Here are the reasons for establishing these six tables respectively and the corresponding relationships among them:

- **Category**: It describes general information of the category, including the level information, which restricts the categorization operations, the number of the child categories, the number of the classified documents, last modified time and etc. These information are the same no matter how many different category names a category has. Therefore in order to avoid redundant storage, we extract these information and store them into the table `Category`.

- **Document**: This is the similar idea to `Category`. We extract the common information of a certain document, including the last modified date, size and etc. and store these information in this table.

- **Category classification**: It is used to describe the relationships among the categories.
Figure 5.4: Database Schema
We use multiple records to represent one to many relationship. Therefore every category can have multiple parent categories by retrieving the records according to Cid (see Fig 5.3). Cname is used to represent the category name, which is the mapping of that category in its parent category. Every category can have more than one category name.

- **Document classification**: This is used to represent the classification of the documents. Every category can have many documents by retrieving the records according to Cid (see Fig 5.3). Every document can be classified to more than one category by retrieving records according to Did (see Fig 5.3). Every document may also have more than one document name.

- **Virtual category classification**: In addition to category classification relationship, we also have virtual category classification relationship. We apply a similar way to define it.

- **URL**: This table is used to store the real location of documents on the disk.

Here we explore this schema in detail and show how it can describe the multiple-categorization hierarchical structure.

**Category** Table 5.1 shows the data stored in Category based on the sample structure showed in Fig 5.5.

- **Cid**: unique identity of a category (Primary key and index)
- **Level**: category level number, which restricts the categorization (see Definition 4.5)
- **Mtime**: Last modified date
- **Cnum**: the number of the subcategories
Figure 5.5: Sample Multiple-categorization hierarchical structure
Table 5.1: Example of Category

<table>
<thead>
<tr>
<th>Cid</th>
<th>Level</th>
<th>Mtime</th>
<th>Cnum</th>
<th>Dnum</th>
<th>Uid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2005-09-30</td>
<td>2</td>
<td>0</td>
<td>root</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2005-09-30</td>
<td>4</td>
<td>0</td>
<td>root</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2005-09-30</td>
<td>1</td>
<td>0</td>
<td>root</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2005-09-30</td>
<td>1</td>
<td>0</td>
<td>root</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2005-09-30</td>
<td>1</td>
<td>0</td>
<td>root</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2005-09-30</td>
<td>1</td>
<td>0</td>
<td>root</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2005-09-30</td>
<td>1</td>
<td>0</td>
<td>root</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>2005-09-30</td>
<td>1</td>
<td>0</td>
<td>root</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>2005-09-30</td>
<td>0</td>
<td>0</td>
<td>root</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>2005-09-30</td>
<td>1</td>
<td>0</td>
<td>root</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>2005-09-30</td>
<td>1</td>
<td>0</td>
<td>root</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>2005-09-30</td>
<td>0</td>
<td>1</td>
<td>root</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>2005-09-30</td>
<td>0</td>
<td>1</td>
<td>root</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>2005-09-30</td>
<td>0</td>
<td>1</td>
<td>root</td>
</tr>
</tbody>
</table>
• **Dnum**: the number of the documents under that category

• **Uid**: user id of the owner

**Category classification** Table 5.2 shows the data stored in **Category classification** based on the sample structure showed in Fig 5.5.

<table>
<thead>
<tr>
<th>Parentid</th>
<th>Cid</th>
<th>Cname</th>
<th>Ctime</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Computers</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>References</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Companies</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>Programming Languages</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>DB</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>Computer Science</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>Education</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>IBM</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>Java</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>JDBC</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>JDBC</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>Dept.</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>13</td>
<td>Canada</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>Universities</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
</tbody>
</table>

• **Parentid**: parent category id (Foreign key and index)

• **Cid**: category id (Foreign key and index)

• **Cname**: category name

• **Ctime**: create time

• **Description**: category description
• The primary key is Cid, Cname and Parentid.

Based on this table, we can get the parent-child relationship among the categories. We can derive the corresponding child categories of a category by retrieving the records according to Parentid. For example, we can get the child categories of category 1 by retrieving Parentid=1. On the other hand, we can derive multiple parent categories of a certain category by retrieving the records according to Cid. For instance, we can get category 12’s parent categories by retrieving records whose Cid=12 (see Fig 5.5 and Table 5.2).

-Virtual category classification Table 5.3 shows the data stored in Virtual category classification based on the sample structure showed in Fig 5.5.

<table>
<thead>
<tr>
<th>Vparentid</th>
<th>Vcid</th>
<th>Vcname</th>
<th>Ctime</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>6</td>
<td>CS</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
</tbody>
</table>

- Vparentid: virtual parent category id (Foreign key and index)
- Vcid: virtual category id (Foreign key and index)
- Vcname: virtual category name
- Ctime: create time
- Description: category description
- The primary key is Vcid, Vcname and Vparentid

Same as those defined above in Category classification, we use multiple records to represent one to many relationship of the virtual category classification. Every category can have more than one virtual child category, which can be retrieved according to Vparentid.
while every category can also have multiple virtual parent categories, which can be got according to \textit{Vcid}.

\textbf{-Document}

<table>
<thead>
<tr>
<th>Did</th>
<th>Mtime</th>
<th>Size</th>
<th>Open</th>
<th>Dncount</th>
<th>Uid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2005-09-30</td>
<td>512</td>
<td>true</td>
<td>1</td>
<td>root</td>
</tr>
<tr>
<td>2</td>
<td>2005-09-30</td>
<td>1024</td>
<td>true</td>
<td>2</td>
<td>root</td>
</tr>
</tbody>
</table>

- \textit{Did}: unique document id (Primary key and index)
- \textit{Mtime}: last modified date
- \textit{Size}: file size
- \textit{Open}: whether open the document to the public or not
- \textit{Dncount}: the number of document names a document has.
- \textit{Uid}: user id of the owner

From the \textbf{Document} table we can see that it defines the general information of a certain document. Every document can only have one record within this table. \textit{Dncount} represents the number of the document names; that is, the number of classifications the document has. This is used to control the existence of a certain document in the system. If \textit{Dncount} is equal to 0 (there is no classification for this document), the document can be erased from the system. This control has an advantage over that in systems such as UNIX file system, where the symbolic link may point to an erased file.
Table 5.5: Example of Document classification

<table>
<thead>
<tr>
<th>Cid</th>
<th>Did</th>
<th>Dname</th>
<th>Ctime</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1</td>
<td>JDBC Tutorial</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>SCS Carleton U.</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>SCS Carleton U.</td>
<td>2005-09-30</td>
<td>...</td>
</tr>
</tbody>
</table>

-Document classification Table 5.5 shows the data stored in Document classification based on the sample structure showed in Fig 5.5.

- Cid: category id (Foreign key and index)
- Did: document id (Foreign key and index)
- Dname: document name
- Description: document description
- The primary key is Did, Dname and Cid

From Table 5.5 we can see that we define one to many relationship by multiple records in this table. Every document can be classified to more than one category and we can get those categories by retrieving the records according to Did. In addition to that, we can also retrieve the documents under a certain category by derive the records according to Cid.

-URL Table 5.5 shows the data stored in URL based on the sample structure showed in Fig 5.5.

- Did: Document id (Foreign key and index)
- Location: absolute path of the location on the disk
- Description: URL description
Generally, every document should have only one real location, such as, in Windows file system, every file has only one unique location on the disk or in directory-based search engines, every webpage should have only one Unique Resource Location (URL). However within a file system, such as UNIX file system, due to the introduction of hard links, a file may have more than one real location on the disk. If we want to erase the file, we have to erase all of its hard links. Therefore in table URL, faced with systems such as UNIX file system, we may have multiple records for a certain document.

From the description above we can see that the schema showed in Fig 5.4 can describe and establish a multiple-categorization hierarchical structure based on the flat tables stored in database.

### 5.4 Basic operations

After discussing the database schema of the category, we examine the basic operations of the category based on that schema. Generally, the core category operations include Add, Remove, Modify and Move.

**Add Category Operation** It is a simple operation, which comprises two steps. First is to establish an category entity, which contains the general information of the category and add the corresponding record to table Category. The second step is to establish the category classification relationship and add the record to table Category classification. For example, in Fig 5.6 we add a category 14 under both category 8 and category 9 with
Figure 5.6: Sample Revised Multiple-categorization hierarchical structure
the category name **Eclipse** and corresponding records are added and revised in the table **Category** and **Category classification** respectively (see Table 5.7, 5.8).

### Table 5.7: Table Category classification after adding category

<table>
<thead>
<tr>
<th>Parentid</th>
<th>Cid</th>
<th>Cname</th>
<th>Ctime</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>8</td>
<td>14</td>
<td>Eclipse</td>
<td>2005-10-01</td>
</tr>
<tr>
<td>ADD</td>
<td>9</td>
<td>14</td>
<td>Eclipse</td>
<td>2005-10-01</td>
</tr>
</tbody>
</table>

### Table 5.8: Table Category after adding category

<table>
<thead>
<tr>
<th>Cid</th>
<th>Level</th>
<th>Mtime</th>
<th>Cnum</th>
<th>Dnum</th>
<th>Uid</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>UPDATE</td>
<td>8</td>
<td>3</td>
<td>2005-10-01</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>UPDATE</td>
<td>9</td>
<td>3</td>
<td>2005-10-01</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>ADD</td>
<td>14</td>
<td>4</td>
<td>2005-10-01</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Remove Category Operation** There are two types of remove operations in our category structure. One is remove the category and the other is remove the category together with its offsprings. The offsprings of a category is defined as the categories, which can be reached from that category; that is all of the descendants of that category. We describe them separately. Suppose \( p_1, p_2, ..., p_i \) are the parent categories of category \( x \) and \( c_1, c_2, ..., c_j \) are the child categories of category \( x \).

- **Remove category without the offspring categories**

Removing category without the offspring categories concerns not only simply removing
the target category, but also the corresponding operations to its offspring categories. We can divide this operation into the following steps:

Suppose we want to remove the category $x$ without its offsprings.

1. Get $x$'s parent list $p_1, p_2, ..., p_i$
2. Get $x$'s children list $c_1, c_2, ..., c_j$
3. Get $x$'s document list $D$
4. For every document of $x$, classify it to $p_1, p_2, ..., p_i$
5. For every child of $x$, add category classification relationship from $p_1, p_2, ..., p_i$ to that child category
6. Remove the category classification relationship from $x$ to $c_1, c_2, ..., c_j$
7. Remove the category classification relationship from $p_1, p_2, ..., p_i$ to $x$
8. Remove $x$

For example, in Fig 5.6, we remove category 10 but keep its offsprings and corresponding records are revised in the table Category, Category classification and Document classification respectively (see Table 5.9, 5.10).

Table 5.9: Table Category classification after removing category without offsprings

<table>
<thead>
<tr>
<th>Parentid</th>
<th>Cid</th>
<th>Cname</th>
<th>Ctime</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOVE</td>
<td>6</td>
<td>10</td>
<td>Dept.</td>
<td>2005-09-30</td>
</tr>
<tr>
<td>REMOVE</td>
<td>10</td>
<td>13</td>
<td>Canada</td>
<td>2005-10-01</td>
</tr>
<tr>
<td>ADD</td>
<td>6</td>
<td>13</td>
<td>Canada</td>
<td>2005-10-01</td>
</tr>
</tbody>
</table>

Remove category together with the offspring categories

Sometimes we need to eliminate a branch of category. In this situation, it's not so convenient to remove the category from the leaves. We can remove the category together with all
of its offspring categories and documents.

Here are the detailed steps:
Suppose we want to remove the category \( x \) with its offsprings.

1. Get \( x \)'s offspring list
2. For every offspring of \( x \) remove the categorization relationship between it and its parents
3. For every offspring of \( x \) remove the documents under that category
4. Remove the offspring
5. Remove the category classification relationship between \( x \) and its parents
6. Remove documents of \( x \)
7. Remove \( x \)

For example, in Fig 5.6, if we remove category 2 together with the offsprings, category 2, 7, 11 and document 2 will be concerned and corresponding changes are made in the tables.

**Move Category Operation** Move category operation is used to move a category together with its offspring branch to a new position. Move operation is based on the operations defined above. However there are something concern us when running the move operation. From our category definition, we can see that there’s a restriction on category level, which guarantees our category structure an acyclic directed graph. When moving a category, there are two main scenarios as shown in Fig 5.7. One is that we move category C from under
category B to category A (B is the category, which has the highest category level number among C’s parent categories, m, n, i represent the level numbers of the categories and m>n>i). In this scenario, we do not encounter the violation of category level restriction to both C and its offspring categories. Therefore, under this scenario, we keep the original category level information of both C and its offspring categories. The other scenario is to move category C from A to B as shown in Fig 5.7 (b) (A is the category, which has the lowest category level number among C’s parent categories, m, n, i represent the category level numbers of the categories and m>n, m>i). Different from the scenario in Fig 5.7 (a), in this situation we have to consider whether there is violation to the category level restriction during the move. We propose a general process procedure, which is listed below:

- Get the difference between m and n, d=m-n
- update the category level information of C and all of its offspring categories by add the difference d
- If get violation to the category level restriction, transfer the category classification to virtual category classification.

Fig 5.8 shows the detailed algorithm of the move operation.

In addition to the category related operations above, we also have document related operations. Document related operations include add, classify and remove. Similar to the operations above the corresponding document related operations manipulate data in table Document, Document Classification and URL.

**Document add**  When adding a document to the category structure, we need to pick up the target category, to which it will be added, and then the corresponding records are generated in table Document, Document Classification and URL.
Figure 5.7: Move operation scenarios
### Algorithm: Move operation

**Input:** Category C to be moved, Target Category T

**Output:** Revised category structure

1. get category level number range of C’s parent categories, \([a, b]\)
2. get the category level number of target category T, \(m\)
3. IF \(m \in [a, b]\) or \(m > b\) THEN
   - add category classification relationship between C and T
   - remove category classification relationship between C and C’s parent categories
4. ELSE
   - get the level difference \(d = l(T) - a\)
   - get C’s offspring categories list \(o\)
   - update both C and its offspring categories’ category level information
   - get C’s subcategories
   - IF subcategory on higher level than C THEN
     - transfer the category classification to virtual category classification from C to that subcategory
   - END IF
   - FOR every category c in \(o\)
     - get c’s subcategories
     - IF subcategory on higher level than c THEN
       - transfer the category classification to virtual category classification from c to that subcategory
     - END IF
   - END FOR
5. END IF

---

Figure 5.8: Algorithm of move operation
**Document classify**  This is used to classify an existing document to more categories in the structure. After classifying the target document to other categories, records are added to table *Document Classification*.

**Document remove**  Since documents in our structure are allowed to be classified to multiple categories, if we remove a document under a certain category, we only remove one mapping of that document (remove records in table *Document Classification*). The document may still exist in the system until the last mapping of the document is removed and then we need to remove records in table *Document, Document Classification* and *URL*. If the document is stored on the local disk, such as UNIX file system, we need to erase the corresponding documents on the disk according to the locations from URL. If the document is on the remote machines, such as those records within directory-based search engines, which only store the URL of the documents, we only need to remove the records in the database.

With the operations above, we can flexibly manipulate our multiple-categorization hierarchical structure. After we have discussed the structure and operations of our category structure, we'd like to investigate how to apply context-based search based on our category structure.
Chapter 6

Context-based search

6.1 Context

As we mentioned in the previous chapter, context-based is not only the most widely used term but also the most ambiguous term in the search domain. Context is defined as the circumstances and conditions which surround an event. In the search domain, context is originally referred to the words, phrases, sentences or paragraphs surround the target keyword. Later on, it is expanded beyond the text-body and concerns the aspects such as personal profiles, types of terminal devices and etc. In my thesis, we define the context as the category hierarchy information of the document. Since we are trying to organize documents into a hierarchical category structure, the hierarchy information of every category does represent the documents placed under that category. For example, category Computers → Programming Languages → Java represents the rough scope and outline of its documents, which are mainly about Java programming languages. Therefore our context-based search is related to the hierarchy information of the category structure and we are trying to wrapping the relevant documents into corresponding categories, from which users could narrow the scope and approach the target according to the hierarchy information. In this chapter we first describe the category related context-based search work, the Context-based Google, based on which we enhanced that method according to our category structure.
6.2 Context-based Google

Context-based Google [1] is the method, in which the system re-organizes the searching results from Google database. By submitting the keyword to Google, we can get a list of relevant documents (see Fig 6.1 (a), the black boxes). However due to the large volume of return data, it is beyond what a user can handle. Since Google has categorized most of its indexed page based on ODP (Open Directory Project) structure dump, its API returns the searching results together with their category information respectively. Therefore Context-based Google can manipulate these information according to its ranking algorithm and return the context-based format of searching result, a list of categories (see Fig 6.1 (b), the black circles). The category hierarchy information displayed as results describes the abstract and brief information about the web pages classified under that category and offers users easier way to narrow the searching space by picking up the corresponding categories, which may contain the desired information under beneath. Furthermore, since the returned documents are wrapped in the corresponding categories, whose number are far less than the number of documents, it’s more likely for users to handle the wrapped results. For example if we want to search the information about the island of Java and submit the keyword of Java, the result may be displayed as followings:

Computers/Programming/Languages/Java

Computers/Internet/Web Design and Development/Hosted Components and Services

Kids and Teens/Computers/Web Page Design/CGI and Programming

Kids Computers Teens/Programming/Languages/Javascript

Regional/Asia/Indonesia/Provinces

(n, m, l, h, x is the number of corresponding target results within the categories.)
6.3 Our enhanced category-related context-based search

6.3.1 Brief introduction

Although Context-based Google improves a lot for the searching results from Google, it still has limitations. In the example above, if we input Java to find the information on Java Island, from Google we will get over 129,000,000 results, which is far beyond a user can handle and our target information is probably listed on the 10th page of the searching result. Even if we apply the Context-based Google, the overwhelming Java-programming-related pages concerning many aspects (i.e. in many different categories) bring us less but still quite a bunch of fuzzy categories. It still takes efforts to locate the category with the target information, especially for hand devices such as PDAs or mobiles which have limited display screens. Context-based Google lists the searching results by expanded categories and we call it a semi-wrapped method. Since we have already wrapped the results
into categories, why don’t we further wrap the categories as well. In the case above, if we wrap our searching results into a subset of our category structure according to the category information carried by the information documents (see Fig 6.1 (c), the black paths), we can get a new category subset, which carries the relevant result. By digging deeper into this category, it’s easier to find our target result. For example, in the use case above, by wrapping our results into categories, we can display the top level categories as follows:

*Computers(n)*

*Kids and Teens(m)*

*Shoppings(l)*

*Regional(x)*

(n, m, l, x is the number of corresponding target results within the categories.)

Due to the limited number of the top level categories, it’s easy to fix the top level domain within the first page. By clicking the category of *Regional* and further on we can easily get our target. Since all the categories within our searching result are the relevant ones, so there will be limited number of subcategories within every category and it will not bring troubles to locate the right subcategory every steps approaching the target.

There is one issue to be concerned. Is it possible to miss our target results by picking up the top level categories? This happens within the directory structure, like Google Directory. However we are benefited from nice characteristics of our multiple-categorization hierarchical structure. Since our enhanced context-based searching is based on a multiple-categorization hierarchical structure, we can approach our target by any possible path, which also gives enough searching-clues to the category related context-based searching. Therefore, there is a better chance of hitting target than using the systems that employ directory structure.

72
6.3.2 Paths

As we mentioned in the previous chapter, path information is one of the features of our structure. In our structure every category or document could have more than one paths from the root. These categories and documents are aware of every possible paths from the root. That makes the categories and documents in our structure can be noted of not only their parent categories, but also all of their ancestors from the root. We apply this feature not only for users’ request for knowing related parents’ and ancestors’ information of a certain category [36], but also for a bottom-up searching solution in our structure. Generally, directory structures such as UNIX file systems applies an up-to-down searching method, in which every node in the structure has to be approached through a tree traversal from the root. However in our structure, we propose an enhanced context-based search, which wraps the target documents into categories and returns with the further wrapped categories according to the path information from the root. It is a bottom-up searching solutions and it is different from most of the directory-based applications, such as UNIX file system, which could only locate a inode through a path but could not retrieve a path according to a inode number. Based on the model and schema we define in the previous chapter, every category or document in our structure could be noted of all the possible paths from the root.

Fig 6.2 shows the detailed algorithm for computing the path information of every category or document. The algorithm is used to compute all the paths between the selected node and the root. We use a stack to store the incomplete path node. Every time we get the top node from the stack and append it with it’s parent-nodes. When the parent node is the root node, we complete a path computation and add that path to the path list. If the parent node is not the root node, we push that node into the stack and update the path information by appending the parent node in front of the original path. By doing so, when the stack is finally empty, we could get a list of paths, which are all the paths between the selected

73
Algorithm of path computing

Input: Category node or document node \( N \)
Output: A list of path information \( P \)

1. Initialize a stack \( S \)
2. Initialize a path list \( P \)
3. Get the input node \( N \)
4. Push \( N \) into \( S \)
5. WHILE \( S \) is not empty
   6. Pop Top node \( T \) from \( S \)
   7. Get \( T \)'s parent nodes list
   8. FOR every \( p \) in parent nodes list
      9. IF \( p \) is not root
         10. Push \( p \) to \( S \) with \( p \) in front of \( T \)'s path
      ELSE
         12. Add a new path to \( P \) with \( p \) in front of \( T \)'s path
     END IF
   14. END FOR
15. END WHILE

Figure 6.2: Algorithm of path computing

category and the top category (see Fig 6.2).

6.3.3 **Enhanced context-based search algorithm**

From Fig 6.1 we can see the differences among the three methods, including the result expressions and hierarchy structures.

- Google searches for keyword-related documents and lists the documents directly as the result. Users have to pick up the target documents from the documents list (see Fig 6.1, black boxes are the search results and grey box is the target).

- *Context-based Google* gets the documents list from Google, wraps them into categories and displays the categories list as the result. Users have to pick up the right category according to the hierarchy information, which represents the documents under it, and get the target documents under beneath (see Fig 6.1, black circles are the search results and the grey box is the target).
In our enhanced context-based search based on multiple-categorization hierarchical structure, we find the documents by keyword-search, wrap them into corresponding categories and further wrap the categories into a subset of the rooted directed acyclic graph according to the multiple path information of the categories. Users can pick up the narrowed category structure to attain the desired targets (see Fig 6.1, categories along the black path are concerned in the result graph and the grey box is the target).

Here are the differences and advantages concern us when designing the algorithm of our enhanced context-based search based on the multiple-categorization hierarchical structure.

- Since our context-based search is based on the multiple-categorization hierarchical structure, in which categories may have multiple paths from the root, we should be able to handle multiple paths rather than single path of the categories.

- In our context-based search, we not only track the category paths, but also refer to the paths from virtual parent-categories.

Fig 6.4 lists the basic steps of our enhanced context-based search based on multiple-categorization hierarchical structure and we list the detailed algorithm in Fig 6.3.

Here is the detailed description of the algorithm:

- First we get the input keyword from the user and do the keyword searching. After that we get a list of relevant documents (see Line 1).

- We wrap the documents into corresponding categories and get a list of categories (see Line 2 to Line 11).

- We collect the paths from every category in that list and prepare for establishing the returned DAG (see Line 12 to Line 15).

- We also append the paths from the virtual-parents along every path in the path list we get in step 3 (see Line 16 to Line 21).
Algorithm of enhanced context-based search

Input: keywords of the target
Output: A rooted directed acyclic graph

1. Keyword search and get a list of related documents D
2. Initialize category list C
3. FOR every document d in D
4. get d's category c
5. IF c is not in C THEN
6. wrap d into c
7. append c to C
8. ELSE
9. wrap d into c
10. END IF
11. END FOR
12. Initialize path list P
13. FOR every c in C
14. get c's path list and append to P
15. END FOR
16. Initialize virtual category path list A
17. FOR every p in P
18. IF get category having virtual parent along the path THEN
19. append that virtual parent’s path list to A
20. END IF
21. END FOR
22. combine P and A to a path list of DAG DAGP
23. establish DAG according to DAGP

*DAG: directed acyclic graph

Figure 6.3: Algorithm of enhanced context-based search
Enhanced context-based search steps

- Combine the paths and establish a rooted directed acyclic graph (see Line 22 to Line 23).
- Output the rooted directed acyclic graph.

For example, if want to find the information on Java island, we submit the keyword Java. The system first retrieves the keyword-relevant records, which include information on Java programming related records, Java island tourism information, Java Coffee and etc. (boxes in Fig 6.1 (c)). Then the system wraps the records into their corresponding categories, such as Computers/Programming/Languages/Java, Computers/Internet/Web Design and Development/Hosted Components and Services, Regional/Asia/Indonesia/Provinces and so on (parent circles of the boxes in Fig 6.1 (c)). By further wrapping these categories from down to up according to their paths, finally the system returns with the limited number of top level categories such as Computers, Kids and Teens, Shoppings and Regional. Now it’s much easier for users to pickup the target information by selecting the top domain.
Chapter 7

Experimental system

After discussing the details of the design and implementation of both the category structure storage and context-based search based on the structure, this chapter describes our experimental system, called Web Category, in detail.

7.1 Roles of the users

As shown in Fig 5.2, there are basically two types of user roles in the system. They are Administrators and Users.

Why do we introduce hierarchical user authorities? As we mentioned in the early chapters, the purpose of our category structure is to take place of the existing directory structure, which is widely used in many applications such as search engines, file systems and etc. Therefore our category structure is not only used for personalization category structure, such as the CASH system does [44]. A hierarchy user access mechanism is a must for a relatively complicated category related application, such as file systems or online posting systems. This can guarantee the management efficiency of the category structure and users holding their own authorization will not mess up the whole category structure, which is used not only by one individual. Since we just establish an experimental system, we only have two types of roles. It can be more complicated in the role hierarchy in some applications such as ODP, which have more than 10 roles.
Administrators  Administrators hold the role of administrations over the whole category manipulations, including the category-related operations such as Add, Edit, Remove, Move and etc. In addition to the general category-related operations, administrators also have the authority to manipulate the information on users account. Fig 7.1 shows a screenshot of the administrator console.

Users  Users are the main anticipants of the system. They post or upload articles into corresponding categories and search for the target information within the system. Fig 7.2 is a screenshot of user console.

7.2 Search types

There are two types of searches within our system: category search and context-based document search. We describe the details of these two search methods in the later section.
On the main search page (see Fig 7.3), there are different options and combinations for search solutions. By selecting the corresponding combination of the options, we submit the search action. The request is sent to the corresponding search controllers and the result will be returned according to the options.

### 7.3 Category manipulation

In this section, we explain the detailed category operations by some practical cases based on our sample category structure (see Fig 7.4). From Fig 7.4 we can see that there are some needs for adjustment over that category structure.

- It's better to move category 11 from category 5 to category 12
- Category 25 under category 15 seems redundant and we want to remove this category but keep it's child category.
We want to classify category 27 to category 13 with the name JDBC.

We want to make a virtual child category under category 16 and it is the mapping of category 7 with the name Computer Science.

We want to upload a document on Java Island and classify it to category 18.

We explain the operations one by one in detail.

**Move Java from Programming to Languages** First of all, find the category of Java within Administrator Console (see Fig 7.5 (a)). Second, click on the move button and select and target, to which the category is moved. (see Fig 7.5 (b)). After we confirm the move action, the category of Java, together with its subcategories JDBC and J2EE will be moved to category of Languages.
Figure 7.4: Sample structure
Remove category of Others  Like what we do in the previous operation, first of all we should find the target category of Others and then click the button of remove (see Fig 7.6 (a) ). There are two ways to remove a category. One is remove the category together with its subcategories and the other is just remove the target category. Since in this case we just want to remove the redundant category of Others, we select the latter one (see Fig 7.6 (b) )

Figure 7.5: Move category

Figure 7.6: Remove category
Edit the hierarchy information of JDBC After finding the target category of JDBC, we click the edit button to edit the detail information of the category (see Fig 7.7(a)). Parents field displays the parent categories and by clicking on the parents we can edit the parent category set (see Fig 7.7(b)) and then find the target parent category, Databases, and add it to the parent category set of JDBC (see Fig 7.7(c)). Now we have JDBC categorized in both Java and Databases.

Figure 7.7: Edit categorization relationship
Add a virtual child category **Computer Science** under **Colleges and Universities**  When establishing virtual child category, the procedure is similar to classify a category to another. Fig 7.8 shows the corresponding changing on the main search page after adding the virtual category classification relationship. From it we can see that **Computer Science**, which is represented by an alias symbol in the beginning of the category name, is a virtual sub-category of **Colleges and Universities** while **Computer Science** is a related category under **Colleges and Universities**. Therefore they can approach each other by corresponding links under them.

![Diagram](image)

**Figure 7.8:** Main category search display after adding logical subcategory

Upload **Java Island** and categorize to **Java**  First, we need to login as a user. We can both upload and post the new document and in this case we apply upload an existing document on the local disk. After selecting the upload document and set the **Title**, we submit the command (see Fig 7.9(a) ). Then we need to select the categories to classify the document (see Fig 7.9(b) ). After picking up the target category, in this case we select **Java**, we classify the uploaded document into that category (sec. Fig 7.9(c) ).

After the operations above, we will get a revised category structure (see Fig 7.10) and the following demonstration searches are based on this revised sample category structure.
Figure 7.9: Upload and categorize documents
Figure 7.10: Revised sample structure
7.4 Multiple search solutions

We offer multiple search solutions within our search system. Users can search by manually digging deeper in our category structure and by submitting keyword to get context-based search. Users can search both documents and categories. Fig 7.3 shows a screenshot of the search main page.

7.4.1 Manually dig deeper

Digging deeper in our category structure is similar to that of both Google Directory and Yahoo. By exploring the category, users find their target information. The differences between our method and that of Google Directory and Yahoo are:

- The exploring is based on our multiple-categorization hierarchical structure and therefore we can have multiple paths to the target categories and meanwhile we can keep the original paths

- We display all the possible paths of the current category during the exploring. That offers users more related search clues. For example in Fig 7.11, when we approach JDBC along the path of Top →Computers →Programming →Languages → Java →JDBC, we can get another possible path of JDBC, Top/Computers/Software/Databases/JDBC and then it can lead us to other related categories such as Databases

---

JDBC
Top/Computers/Programming/Languages/Java/JDBC/
Top/Computers/Software/Databases/JDBC/

Figure 7.11: Manual search
7.4.2 Context-based searches

In our system, we supply multiple context-bases search solutions. By selecting the search object (documents or categories), the display format (in expanded categories or wrapped categories), the searching space constraints (whether under current category) and inputting the keyword, users get the context-based search results according to their options. We use two examples to illustrate the context-based searches within our system.

Search for *Java* and display the result in wrapped category  We input *Java* and select the wrapped category method (see Fig 7.12). After submitting the search query we will get the wrapped result (see Fig 7.13). We can dig deeper within the new subset of the category structure, which contains the categories with the target documents.

![Figure 7.12: Search for Java and display the result in wrapped category](image)

Search for *SCS Carleton U.* under Education  As what we mentioned in the early chapter of the thesis. If we want to find the information on *SCS Carleton U.* under the category of...
Education in Google Directory, we will get no result. Although we can manually approach
our target from the category of Education, which has alias linker within its subcategory
to the category under the branch of Computers, it does not track the alias linker information during the search. However in our system, we not only track all the possible paths of
the categories, but also take the sources of virtual parent-categories as reference. Fig 7.14 shows the detailed algorithm of our search under a certain category.

![Algorithm: Search under a certain category](image)

**Algorithm: Search under a certain category**

**Input:** keyword, current category c

**Output:** result category list

1. get the relevant document list D
2. get the current category c
3. wrap D into category list of C
4. initialize a result list R
5. FOR every category cat in C
6. get path list p of cat
7. IF c in any path within p THEN
8.    add cat to R
9. ELSE
10.    get categories along every path within p
11.   IF the category has input from virtual parent category A THEN
12.     IF c in A's path THEN
13.       add cat to R
14.   END IF
15. END IF
16. END IF
17. END FOR

Figure 7.14: Algorithm: Search under a certain category

### 7.4.3 Summary

From the sample search solutions above we can see that our experimental system based on
multiple-categorization hierarchical structure overcomes the problems we propose in the chapter 1.

- Manually digging deeper operation offers more flexible and more clue-sensitive method
  than the existing directory structure by providing multiple paths of a category, which
  offers users more flexibility to switch among the related categories to approach the target.
• Context-based searching in wrapped category provides limited number of results for users to pick up. If within a web searching system, it is much easier for users, who are interested in the information on Java Island, to pick up the target information from overwhelming Java programming related web pages.

• Searching under a certain category retrieves the paths information from virtual parent categories along the result category paths and take for a reference. Therefore even if the document is not categorized under a certain category, but under the virtual child category of that category, it can be retrieved.

7.5 Experiment and comparison

After doing the sample demonstration of our prototype system, we establish an experiment to map both UNIX file system and ODP structure dump into our category structure separately. By doing so, we can check the prevalent-usage-possibility of our structure.

7.5.1 UNIX file system mapping

Since the whole file system has more than 15 levels' directory, we simplify it to map only 5 level of the file system. In order to handle the UNIX file system based on the multiple-categorization hierarchical structure, we have to first map it into our multiple-categorization hierarchical structure. Here are the related information on the file system we handle during the mapping.

• Directories: we handle these as categories and map them into our category related tables.

• Files: we handle these as documents and map them into our document related table.

• Links: there are links to both directories and files within the file system and we handle them separately. If the link is to a file, then the target file will be classified
Algorithm: UNIX file system mapping

/*Load File System*/

**Input:** root directory

**Output:** multiple-categorization hierarchical structure

1. get the directory F
2. IF F can read THEN
3. get F's sublist of F-list
4. FOR every f in F-list
5. IF f is a file THEN
6. classify f to F
7. END IF
8. IF f is a directory THEN
9. classify f to F
10. call Load File System with the input of f
11. END IF
12. END FOR
13. END IF

/*Map Links*/

**Input:** root directory

**Output:** multiple-categorization hierarchical structure with links

14. get the directory L
15. IF L can read THEN
16. get L's sublist of L-list
17. FOR every l in L-list
18. IF l is a link to a file THEN
19. get the target file and classify under L
20. END IF
21. IF l is a link to a directory THEN
22. get the target directory t
23. IF t in a lower level than L THEN
24. classify t to L
25. ELSE
26. virtually classify t to L
27. END IF
28. END IF
29. IF l is a directory THEN
30. call Map Links with the input of l
31. END IF
32. END FOR
33. END IF

Figure 7.15: Algorithm: UNIX file system mapping
under where the link is from. If the link is to a category, then we have two scenarios. First, if the target category is in a lower level than the source category, the target category will be classified under the source one. Second, if the target category is in a higher level, then that category will be virtually classified under the source category.

We take twice traversal of the file system. First, we load the file system into our database and then in the second round, we handle the link operations. Fig 7.15 shows the detailed algorithm of UNIX file system mapping.

Mapping between ODP RDF structure and links on the web page

Figure 7.16: Mapping between ODP RDF structure and links on the web page

7.5.2 ODP structure dump mapping

ODP structure dump is defined in RDF [43], which is an XML format document. Generally, we apply DOM [28] or SAX [42] to parse XML document. However the size of the structure dump file is over 50M and we cannot parse it with DOM (DOM loads the whole

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
document into the memory before processing, which is memory consuming). Therefore we apply SAX to handle the mapping job. There are more than 20 tags are defined within the ODP structure dump while we concern only part of them. They are: Alias, Target, Topic, narrow, narrow1, narrow2, symbolic, symbolic1, symbolic2. For example, in Fig 7.16 we have a segment of RDF structure, in which the Topic r:id="Top/Arts" represents the current category and the meta information under element of narrow, symbolic and etc. represents the child categories. The elements within Alias are mapped into our child categories or virtual categories with respect to their level information. We implement the SAX element handler to these tags respectively (see Fig 7.17).

<table>
<thead>
<tr>
<th>ODP mapping algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Top tag</td>
</tr>
<tr>
<td><strong>Output:</strong> multiple-categorization hierarchical structure</td>
</tr>
</tbody>
</table>

1. get the Tag Name
2. **IF** Tag Name is Topic **THEN**
   3. save the Topic ID P
   4. **IF** P is not established **THEN**
      5. insert a new category information
   6. **ENDIF**
3. **ELSE IF** Tag Name is child-category related tags **THEN**
   4. get the resources ID C
   5. **IF** C is not established **THEN**
      6. insert a new category information
      7. insert a new categorization relationship information between P and C
   8. **ENDIF**
4. **ELSE IF** Tag Name is alias-linker related tags **THEN**
   5. get the resources ID A
   6. **IF** A is not established **THEN**
      7. insert a new category information
      8. **IF** the target category T is not established **THEN**
         9. insert a new target category T
      10. **ENDIF**
     11. **IF** T is in a higher level than A **THEN**
        12. insert a virtual categorization relationship between A and T
     13. **ELSE**
        14. insert a categorization relationship between A and T
     15. **ENDIF**
   16. **ENDIF**
26. **ENDIF**

Figure 7.17: ODP mapping algorithm
After mapping the ODP structure dump into our database schema, the corresponding links and alias links (see Fig 7.18, the alias link has been transferred to multiple categorization) in the dump are handled and we can apply features of our category structure to the ODP structure dump.

![Figure 7.18: Category of Games](image)

7.5.3 Feature-based comparison

First, we compare the features of our Web Category system with those of CASH system [44], Google, Context-based google, file systems, posting systems and other directory based application to show that multiple-categorization hierarchical structure based system is more flexible and natural to organize, manage, maintain, and search for information.

Web Category VS. CASH Our research is in a similar way to CASH system. However CASH system still have some limitations and drawbacks, which are mentioned in the early chapter, and our system has overcome these drawbacks and limitations.

- CASH is platform-dependent while our Web Category is a web interface, which can be applied on any platforms.

- CASH does not handle the links within the file systems when mapping while our Web Category system fully handle these supplementary hierarchy information in file systems, directory-based search engines and etc. Fig 7.19 shows the top level after mapping server’s file system into our system. We consider two categories, www and
ygao. *www* is originally a symbolic link under the top directory of the file system and it points to `/usr5/restaurantsystem/`. In this case, based on our definition and implementation, we classify this directory to the top directory and with the name of *www*. By clicking *www*, we approach that category, which has two possible paths listed above (see Fig 7.21). *ygao* on the other hand is not a category or link under the top directory of the file system. It is the established by a virtual child category mapping from `/usr4/ygao/`. By clicking it, we approach that corresponding category, under which there is a virtual child category *@root*, the symmetric category of the related category *ygao* (see Fig 7.20).

![Web Category](image)

**Figure 7.19: Web Category based Unix File System**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 7.20: Category of ygao
- CASH is based on command line, which is a little bit boring and complicated for unprofessional users while our system is based on graphical interface. Users can manipulate or search the categories and documents by some simple click operations and do not need to memorize the commands.

![Web Category](image)

Figure 7.21: Category of www

- Although CASH is based on category structure, its categories do not contain the whole paths information. In our system, all the possible paths information of a certain category is recorded and shown during the search. For example, in Fig 7.21, we can have two possible paths for that category.

- CASH offers lots of category manipulating operations, which are also implemented
in our system but in a graphical way. However CASH lacks flexible and efficient search solutions especially for the searching cases with incomplete keyword. In our system, we provide manual search, context-based search and enhanced context-based search, which can cover the general usage of information searching. For example, if a user wants to find some document and he can only provides the incomplete keyword like *lib*. Within CASH system we may use the command *ls -n=*lib* and that returns more than one thousand related results, which is a disaster for user to filter the fuzzy results. By our system we can submit the keyword of +lib+ and apply either context-based or enhanced context-based search. After submitting the request
we will get the results wrapped in the corresponding categories and each category are symbolized with its path information, which is the semantic representation of that category (see Fig 7.22). It’s much easier for users to pickup the desired documents. If it returns a lot of result categories, we can also apply the enhanced context-based search, which wraps the categories and then users can perform the digging deeper operation to approach their target.

**Web Category VS. other directory based applications** Table 7.1, 7.2, 7.3 show the comparisons between our Web Category system and other systems from four aspects:

<table>
<thead>
<tr>
<th></th>
<th>Our Web Category</th>
<th>File System</th>
<th>Google</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Categorization</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Keep all paths</td>
<td>Yes</td>
<td>Single path</td>
<td>Single path</td>
</tr>
<tr>
<td>Refer paths from virtual parent-category</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>Search result</td>
<td>multiple options</td>
<td>related doc list</td>
<td>related records list</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Context-based Google</th>
<th>Library of Congress</th>
<th>Java Developer Forum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Categorization</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Keep all paths</td>
<td>Single path</td>
<td>N/A</td>
<td>Single path</td>
</tr>
<tr>
<td>Refer paths from virtual parent-category</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Search result</td>
<td>Wrapped result list</td>
<td>related record list</td>
<td>related record list</td>
</tr>
</tbody>
</table>
Table 7.3: Comparison (continue2)

<table>
<thead>
<tr>
<th></th>
<th>Email management</th>
<th>Bookmark console</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Categorization</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Keep all paths</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Refer paths from virtual</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>parent-category</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search result</td>
<td>related</td>
<td>related</td>
</tr>
<tr>
<td></td>
<td>record list</td>
<td>record list</td>
</tr>
</tbody>
</table>

- Multiple Categorization: it offers users more natural and flexible way to handle the information or document.

- Keep all paths: it offers more related descriptions of information or document for both implicit and explicit search. For explicit search, the possible paths can be displayed to the users for their reference to get related information or approach the target one. As for the implicit search, on the other hand, is for the machine processing.

- Refer paths from virtual parent-categories: it offers more supplementary descriptions of the information or documents during the search.

- Search result: it is an important part for information searching. No matter how the information within a system is organized or indexed, the purpose of these systems is to display limited number of most relevant results for users to pick up.

The first aspect reflects the natural and flexible way of information classification and the other three aspects illustrate the efficiency and accuracy of information retrieval. By comparing our system with other applications according to these four aspects, we can see that the system based on our multiple-categorization hierarchical structure offers more flexible and efficient way to classify and retrieve information.
7.5.4 Performance-based comparison

After discussing the feature-based comparisons between the system based on our category structure and other directory-based applications, here we take a look at the actual performance of our system and make the comparison with those directory-based applications.

As we mention in the previous chapter that directory-based applications apply tree traversal to locate a certain directory or file, such as that in UNIX file system. However in our system based on category structure, we utilize database to store the category and document nodes and apply index on those nodes. Therefore due to the advantages of index, which applies B-Tree search, we should have better performance than those directory-based applications. There is one issue should be concerned. We apply an enhanced context-based search based on the searching results, which is an overhead in addition to the conventional search methods. Does it affect a lot to our performance? With these issues and puzzles, we establish an experiment to test the performance and do the comparisons.

The test is based on our mapped UNIX file system, which covers 5 levels of the file system, and with 2270 categories and 3467 documents in our database. We perform both tree traversal search and enhanced context-based search based on 5-level-UNIX file system. We pickup the sample search keywords to gain an ascending number of results to check the different performances based on the sample data. Table 7.4 shows the test result and Fig 7.23 shows the corresponding diagram.

<table>
<thead>
<tr>
<th>The number of result</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tree traversal:</td>
</tr>
<tr>
<td>1</td>
<td>7369</td>
</tr>
<tr>
<td></td>
<td>enhanced context-based: 3229</td>
</tr>
<tr>
<td>31</td>
<td>8674</td>
</tr>
<tr>
<td></td>
<td>enhanced context-based: 4675</td>
</tr>
<tr>
<td>113</td>
<td>8802</td>
</tr>
<tr>
<td></td>
<td>enhanced context-based: 11836</td>
</tr>
<tr>
<td>675</td>
<td>8567</td>
</tr>
<tr>
<td></td>
<td>enhanced context-based: 44763</td>
</tr>
<tr>
<td>1453</td>
<td>8761</td>
</tr>
<tr>
<td></td>
<td>enhanced context-based: 90894</td>
</tr>
</tbody>
</table>

Table 7.4: Search performance comparison
From the diagram we can see that with the rising of result number, there is no obviously changes on time consuming for tree traversal method. That is because, every time it begins from the root and performs the traversal till the leaves before returning the results, no matter how many results will be returned in the end. On the other hand, when performing enhanced context-based search, there's an obvious rising tendency on the time consuming because the more results are returned, the more wrappings will be done. However due to the database index searching advantages, when the number of search result is not so high, i.e. searching by a complete and unambiguous keyword, the performance of our structure is no worse than the file system even if we have overhead from database operations. If the number of result is very huge, which leads to rapid growth of time consuming, based on the discussion in the previous section, our method still offers more flexible and simpler way for users to pick up the target from the results.
Chapter 8
Conclusion remarks and future work

In order to organize and search information in a more flexible and natural way, we describe a novel category structure, called multiple-categorization hierarchical structure, which allows categories and documents to be multiply categorized. Based on the nice features of this structure, we propose an enhanced context-based search method, which can retrieve target information in a more flexible and accurate way. After discussing the novel category structure and the enhanced context-based search method, we describe our experimental system, Web Category, which is based on the multiple-categorization hierarchical structure. By both UNIX file system and ODP structure dump mapping experiments, we show that our experimental system offers more flexible and natural way to organize and search for information than other directory based applications. Since our work is mainly an extension to CASH system [44], we compare our work with CASH system throughout the thesis and here is the list to make a summary:

- CASH [44] is short of adequate survey and research work on the existing systems and related work while we have done some amendments and research on the related works and systems from both academic community and industry.

- CASH [44] puts little efforts on formalizing the general concepts and properties of the category structure while we have given out the detailed definitions of concepts within the category structure.
• CASH [44] is programmed based on the extension of a shell on the file system, which leads it to platform-dependent, while we implement the system by web application framework, which is independent and can be applied on all most any kinds of platforms.

• CASH [44] still possesses some drawbacks from UNIX file system, such as it could locate the target via an absolute path but could not get all the paths information via a target. This restricts the bottom-up searching method and could not fulfil users’ request for being noted of more information on a target [36].

• CASH [44] is short of search power, which is greatly enhanced in our work

• Although CASH [44] proposed the idea to attain efficient management via introducing supplementary databases system based on the file system, it is poorly modeled and designed and does not attain the full use of database advantages. Therefore, in our work, we re-design the whole schema and structure to attain maximized database management efficiency.

• CASH [44] incited a nice idea and structure, however it does not thoroughly prove the advantages of the structure and system while we have established experiments in our work and compare the category structure with other structure from both features and performance.

Those are also the contributions of our work. Here are the corresponding pros and cons of our work:

-PROS

• Applying database to manage the categories and documents and taking advantages of nice indexing and searching features of databases over conventional tree traversal method in directory-based applications
• Platform-independent
• More natural and flexible to classify documents
• More efficient and accurate to search for information
• Friendly interface

- CONS

• Huge and complicated hierarchy structure may cause more overhead on computing and affect the searching speed.

• Applying DBMS to manage the structure seems cumbersome and the overhead within DBMS may also affect the efficiency of the system.

Based on these two drawbacks we propose two openings for future research. We will go on working on these opening questions:

• Are there any further advanced paths computing alternatives to improve the efficiency of computing on the fly, such as parallel computing?

• We may also apply some embedded database engines instead of cumbersome DBMS and have these engines specific for our usage. That may reduce some redundant functions and overhead, which may affect the efficiency. One of the candidate database engines will be Berkeley DB.
Bibliography


107

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.


